UNIVERSIDAD CARLOS III DE MADRID

PROYECTO FIN DE CARRERA DE INGENIERÍA TÉCNICA EN

INFORMÁTICA DE GESTIÓN

USO DE PATRONES DE PRODUCTO EN

METODOLOGÍAS ÁGILES

AUTORA: ***Patricia Muela Gordillo***

DIRIGIDO POR: **María Isabel Sánchez Segura**

JULIO 2010

AGRADECIMIENTOS

Al fin llegó el momento de escribir los agradecimientos, uno de los instantes más deseados y que creí que nunca llegaría.

En primer lugar, quiero agradecer a mi familia, mis padres, mis hermanos José, Lourdes y Reyes, cuñados y sobrinos por tanto apoyo a lo largo de toda mi vida, pero en especial en los años que he pasado en la carrera.

A José, por tantos momentos de nervios, estrés, subidas y bajadas emocionales que tanto has aguantado, porque sin ti no hubiera podido llegar donde estoy. Gracias, porque en situaciones que me superaban siempre he tenido tus palabras de apoyo.

A mis becarios, ex–becarios y personal de biblioteca, con los que tantas horas he pasado, porque para mí ya no son compañeros, sino amigos, con los que he compartido la vida en estos años de becaría.

Pero especialmente, este proyecto va dedicado a mis padres, Antonio y Dolores. Nunca podré agradeceros tanto apoyo y cariño, en momentos que no han sido fáciles, pero que sin vosotros no hubiera conseguido llegar a ser la persona que soy.

No puedo decir otra cosa que GRACIAS POR TODO.

INDICE

HOJA DE ACTIVIDAD DEL PROYECTO

REGISTRO DE DATOS DEL PROYECTO
NOMBRE: Uso de patrones de producto en Metodologías Ágiles
FECHA: 22 de Julio del 2010

		VALORES
1	¿Cuántos documentos se han creado?	8
2	¿Cuántos documentos están pendientes de revisar?	0
3	¿Cuántos documentos se han revisado?	8
4	¿Cuántos documentos se han finalizado?	8
5	¿Cuántas versiones se han creado del documento? (número más alto)	4
6	¿Cuántas reuniones se han hecho con el jefe de proyecto (tutor)?	10
7	¿Cuál es el objetivo del proyecto?	El proyecto trata del uso de patrones de producto en métodos de desarrollo ágil, concretamente aplicados a la metodología XP. Con ello lo que se pretende es observar si existe una mejora sustancial en cuanto al tiempo de desarrollo de los productos de software al incorporar el uso de patrones de producto para el desarrollo de los mismos.
8	Fecha de inicio del proyecto	1/9/2009
9	Fecha de finalización del proyecto	22/7/2010

1. INTRODUCCIÓN

Al menos tres importantes estudios del estado de la ingeniería del software realizados a mediados de los noventa, indican que la mayoría de los equipos que desarrollan software no cuentan con metodologías o procesos que permitan cumplir con las expectativas de calidad, tiempo y alcance a costos razonables. Menos del 10% de los proyectos de software son entregados cumpliendo con los recursos inicialmente asignados y en la fecha previamente establecida. El 40% de los proyectos se extendieron en el 67% del tiempo inicialmente establecido, el 30% se extendieron en un 127% del presupuesto inicialmente establecido. Por este motivo, al utilizar metodologías tradicionales, se puede esperar que la mitad de los proyectos se retrasen y un tercio de ellos se excedan en presupuesto.(Machado, F. , 2001)

Esto se debe, en gran parte, a la utilización de las metodologías tradicionales de desarrollo que no se ajustan a la velocidad y constancia de los cambios de la actualidad. Los mercados de hoy, son altamente dinámicos y competitivos, lo que hace que las reglas de negocio tengan que cambiar continuamente para mantener a las organizaciones saludables y competentes. A partir de esta realidad, los equipos de desarrollo deben entregar funcionalidad rápidamente y de manera continua, brindando valor para el presente, pero sabiendo que este valor se puede perder a corto plazo.

La tecnología, a menos que sea el fin de una organización en sí misma, se utiliza fundamentalmente para disminuir costos o aumentar la productividad. Hasta la década de los noventa, las organizaciones competían principalmente en sus habilidades de ofrecer productos y servicios, aquellos que utilizaban una tecnología de soporte cara y compleja de mantener en ese entonces, tenía una ventaja competitiva. Pero el mundo está cambiando, existen soluciones tecnológicas cada vez más económicas y sencillas, accesibles para cualquier organización, las cuales compiten en agilidad o velocidad para ejecutar cualquier cambio.

Uno de los supuestos universales de la ingeniería del software es que los costos de cambiar un programa crecen exponencialmente a lo largo del tiempo. Las metodologías tradicionales tales como el modelo en cascada y en espiral, están basadas en la validez de este supuesto, y por consiguiente, están diseñadas para determinar todos los requerimientos en las primeras etapas y no están preparadas para afrontar los cambios constantes.

Las metodologías tradicionales se consideran “pesadas” ya que están basadas en complejos procedimientos y reglas que hacen que los integrantes del grupo tengan que dedicar gran parte de su tiempo en el cumplimiento del proceso, corriendo el riesgo de perder el foco de atención en el producto a desarrollar.

Por ello surgen las **Metodologías Ágiles.**

2. METODOLOGÍAS ÁGILES EN EL DESARROLLO DE SOFTWARE

Existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Propuestas más tradicionales, efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Una posible mejora es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados. Sin embargo, el resultado final sería un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto. Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto software. Ésta es la filosofía de las metodologías ágiles. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. (Letelier, P. Panadés, M.C. y Canós, J.)

Hasta hace poco el proceso de desarrollo llevaba asociada un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos). Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. . (Letelier, P. Panadés, M.C. y Canós, J.)

En este escenario, las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas para proyectos pequeños, las metodologías ágiles constituyen una solución a medida para ese entorno, aportando una elevada simplificación que, a pesar de ello, no renuncia a las prácticas esenciales para asegurar la calidad del producto.

Las metodologías ágiles son sin duda uno de los temas recientes en ingeniería de software que están acaparando gran interés.

La curiosidad que siente la mayor parte de ingenieros de software, sobre las metodologías ágiles hace prever una fuerte proyección industrial. Las características de los proyectos para los cuales las metodologías ágiles han sido especialmente pensadas se ajustan a un amplio rango de proyectos industriales de desarrollo de software; aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, y/o basados en nuevas tecnologías.

En febrero de 2001, tras una reunión celebrada en Utah-EEUU, nace el término "ágil" aplicado al desarrollo de software. En esta reunión participan un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto. . (Letelier, P. Panadés, M.C. y Canós, J.)

Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

Tras esta reunión se creó The Agile Alliance, una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida fue el Manifiesto Ágil, un documento que resume la filosofía "ágil".

Según el Manifiesto se valora:

- Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas. La gente es el principal factor de éxito de un proyecto software. Es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.

- Desarrollar software que funciona más que conseguir una buena documentación. La regla a seguir es no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante. Estos documentos deben ser cortos y centrarse en lo fundamental.

- La colaboración con el cliente más que la negociación de un contrato. Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.

- Responder a los cambios más que seguir estrictamente un plan. La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

Los valores anteriores inspiran los doce principios del manifiesto. Son características que diferencian un proceso ágil de uno tradicional. Los dos primeros principios son generales y resumen gran parte del espíritu ágil. El resto tienen que ver con el proceso a seguir y con el

equipo de desarrollo, en cuanto metas a seguir y organización del mismo. Los principios son:

I. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.

II. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.

III. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.

IV. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.

V. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.

VI. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.

VII. El software que funciona es la medida principal de progreso.

VIII. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.

IX. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.

X. La simplicidad es esencial.

XI. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.

XII. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

(Letelier, P. Panadés, M.C. y Canós, J.)

2.2 Comparación entre metodologías ágiles y tradicionales

Las diferencias entre las metodologías tradicionales y las metodologías ágiles afectan no sólo al proceso en sí, sino también al contexto del equipo así como a su organización.

A continuación en la tabla 1 recogemos esquemáticamente las principales diferencias de las metodologías ágiles con respecto a las tradicionales ("no agiles").

METODOLOGÍAS ÁGILES	METODOLOGÍAS TRADICIONALES
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/nomas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (menos de 10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Tabla 1: Diferencias entre metodologías ágiles y tradicionales

Las metodologías de desarrollo de software tradicionales (ciclo de vida en cascada, evolutivo, en espiral, iterativo, etc.) aparecen, como pesadas y poco eficientes. La crítica más frecuente a estas metodologías "clásicas" es que son demasiado burocráticas. Hay tanto que hacer para seguir la metodología que, a veces, el ritmo entero del desarrollo se retarda. Como respuesta a esto, se ha visto en los últimos tiempos el surgimiento de "Metodologías Ágiles". Estos nuevos métodos buscan un punto medio entre la ausencia de procesos y el abuso de los mismos, proponiendo un proceso cuyo esfuerzo valga la pena. (Josskowicz, 2008)

Los métodos ágiles cambian significativamente algunos de los énfasis de las metodologías "clásicas":

- Los métodos ágiles son adaptables en lugar de predictivos. Los métodos "clásicos" tienden a intentar planear una gran parte del proceso del software en gran detalle para un plazo largo de tiempo. Esto funciona bien hasta que las cosas cambian. Así que su naturaleza es resistirse al cambio. Para los métodos ágiles, no obstante, el cambio es bienvenido. Intentan ser procesos que se adaptan y crecen en el cambio.

- Los métodos ágiles son orientados a la gente y no orientados al proceso. El objetivo de los métodos "clásicos" es definir un proceso que funcionará bien independientemente de quien lo utilice. Los métodos ágiles afirman que ningún proceso podrá nunca maquillar las habilidades del equipo de desarrollo, de modo que el papel del proceso es apoyar al equipo de desarrollo en su trabajo.

(Letelier, P. Panadés, M.C. y Canós, J.)

3. UNA INTRODUCCIÓN A LA PROGRAMACIÓN EXTREMA (EXTREME PROGRAMMING, XP)

"Extreme programming" o "Programación extrema" es una de las llamadas metodologías ágiles de desarrollo de software más exitosas de los tiempos recientes. La metodología propuesta en XP está diseñada para entregar el software que los clientes necesitan en el momento en el que lo necesitan.

Forma parte de los métodos ágiles que centran sus prioridades en las personas, no en los procesos.(Anaya, A.)

XP alienta a los desarrolladores a responder a los requerimientos cambiantes de los clientes, aún en fases tardías del ciclo de vida del desarrollo.

La metodología también enfatiza el trabajo en equipo. Tanto gerentes como clientes y desarrolladores son partes del mismo equipo dedicado a entregar software de calidad.

Este método reciente de desarrollo orientado a objetos fue descrito originalmente por Kent Beck sobre finales de 1990s. En la actualidad está generando muchos adeptos a través de Internet y tiene cada vez una mayor presencia como un método alternativo de desarrollo frente a los métodos más clásicos. Se basa principalmente en la simplicidad, la comunicación e interacción permanente con el cliente (comprobación de requisitos constante) y en el "pair-programming", que es la técnica de programación por parejas donde uno de los programadores escribe el código y el otro lo prueba, y después se cambian los papeles. De esta forma ya desde el principio se van probando los programas en cuanto a cumplimiento de requisitos como la funcionalidad. La simplificación de los protocolos de comunicación entre las diferentes fases y la inmediatez del desarrollo lo convierten en una metodología muy "rápida" de desarrollo.(Álvarez, J.R. y Arias, M.)

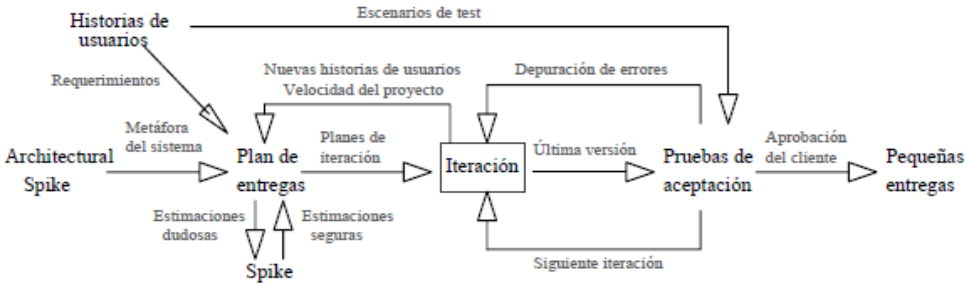


Figura 1: Etapas de extreme programming

Sus características son:

- Permite introducir nuevos requisitos o cambiar los anteriores de un modo dinámico.
- Publica pronto versiones que implementan parte de los requisitos.
- Es adecuado para proyectos pequeños y medianos.
- También es adecuado para proyectos con alto riesgo.
- Su ciclo de vida es iterativo e incremental. Cada iteración dura ente una y tres semanas.
- Un defecto de extreme programming es que necesita una continua interacción con el usuario, y no con cualquiera, sino con alguien que conozca la empresa y sus necesidades.
- Otra crítica que ha hecho de esta metodología es que no produce demasiada documentación acerca del diseño o la planificación.

(Álvarez, J.R. y Arias, M.)

El objetivo de XP son grupos pequeños y medianos de construcción de software en donde los requisitos aún son muy ambiguos, cambian rápidamente o son de alto riesgo. XP busca la satisfacción del cliente tratando de mantener durante todo el tiempo su confianza en el producto. Además, sugiere que el lugar de trabajo sea una sala amplia, si es posible sin divisiones (en el centro los programadores, en la periferia los equipos individuales). Una ventaja del espacio abierto es el incremento en la comunicación y el proporcionar una agenda dinámica en el entorno de cada proyecto.

«Todo en el software cambia. Los requisitos cambian. El diseño cambia. El negocio cambia. La tecnología cambia. El equipo cambia. Los miembros del equipo cambian. El problema no es el cambio en sí mismo, puesto que sabemos que el cambio va a suceder; el problema es la incapacidad de adaptarnos a dicho cambio cuando éste tiene lugar.»
Kent Beck .(Fernández, G. , 2002)

Para algunas personas, XP parece algo de sentido común. Entonces ¿Por qué "extrema" en su nombre? XP toma principios y prácticas de sentido común a niveles extremos.

- Si las revisiones de código son buenas, se revisará el código todo el tiempo (programación en parejas).
- Si las pruebas son buenas, todos probarán durante todo el tiempo (prueba unitaria), incluso los clientes (prueba funcional).

- Si el diseño es bueno, se hará que forme parte de la ocupación diaria de todos (recodificación).

- Si la simplicidad es buena, siempre se dejará el sistema con el diseño más sencillo que soporte su funcionalidad actual (la cosa más sencilla que probablemente pudiese funcionar)

- Si la arquitectura es importante, todos trabajarán definiendo y refinando la arquitectura todo el tiempo (metáfora).

- Si las pruebas de integración son buenas, se harán iteraciones realmente cortas, segundos, minutos y horas, y no semanas, meses y años (el juego de la planificación).

XP hace dos tipos de promesas:

- A los programadores, XP les promete que serán capaces de trabajar en cosas que realmente importan, cada día. No tendrán que enfrentarse solos a situaciones aterradoras. Podrán aprovechar todas sus energías para hacer que su sistema tenga éxito. Tomarán las decisiones que puedan tomar mejor, y no tomarán aquellas para las que no sean los mejores preparados.

- A los clientes y directivos, XP les promete que obtendrán el mayor valor posible de cada semana de programación. Cada pocas semanas podrán ver progresos concretos en las metas que les importan. Podrán cambiar la dirección del proyecto a mitad del desarrollo, sin incurrir en costes excesivos.

Una pequeña estimación de la inclinación de los desarrolladores en la utilización de XP podría ser la siguiente:

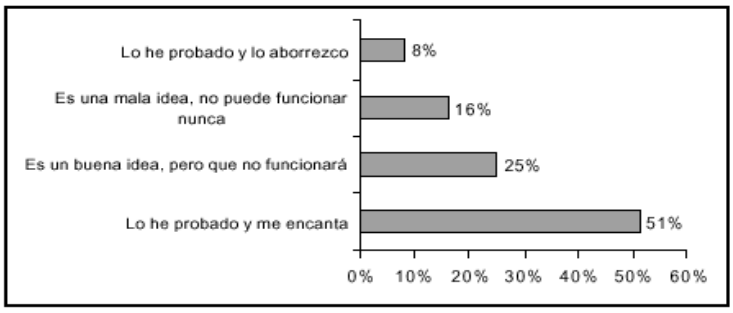


Figura 2: Estimación

XP propone reducir el riesgo del proyecto, mejorando la sensibilidad a los cambios del negocio, mejorando la productividad a lo largo de la vida de un sistema; y añade diversión a la construcción del software en equipo, todo al mismo tiempo. (Beck, 2000)

Extreme programming se diferencia de otros métodos en:

- Su inmediata, concreta y continua realimentación de los ciclos cortos.
- Su enfoque de planificación incremental, que rápidamente plantea un plan global que se espera que evolucione a lo largo de la vida del proyecto.
- Su capacidad para programar de forma flexible la implementación de funcionalidades, respondiendo a las necesidades cambiantes de los negocios.
- Su confianza en las pruebas automatizadas, escritas por los programadores y los clientes para controlar el progreso del desarrollo, para permitir la evolución del sistema y captar los defectos lo antes posible.
- Su confianza en la comunicación oral, las pruebas y el código fuente para comunicar la estructura e intención del sistema.
- Su confianza en el proceso de diseño evolutivo que perdura mientras perdure el sistema.
- Su confianza en la colaboración estrecha entre programadores con habilidades normales.
- Su confianza en las prácticas que funcionan tanto con los instintos a corto plazo de los programadores como los intereses a largo plazo del proyecto.

XP es una disciplina de desarrollo del software. Es una disciplina porque hay cosas que tienes que hacer para aplicar XP. No se puede escoger si se escribirán o no pruebas, si no se hace, no se es extremo. (De Seta, L., 2010)

Extreme programming está diseñado para trabajar con proyectos en los que puedan llevarse a cabo con equipos de dos a diez programadores, que no estén fuertemente condicionados por el entorno de computación existente, y en donde un trabajo razonable de ejecución de pruebas puede hacerse en una fracción de un día.

Ninguna de las ideas de XP es nueva. Muchas de ellas son tan viejas como la programación. Hay un sentido en el que XP es conservadora, pues todas sus técnicas han sido probadas durante décadas (la estrategia de implementación) o siglos (la estrategia de gestión).(Beck, 2000)

La novedad de XP está en poner todas estas prácticas bajo un mismo paraguas, asegurar que todas son practicadas tan minuciosamente como sea posible y asegurar que las prácticas se apoyan unas a otras en el mayor grado posible.

4. EL PROBLEMA BÁSICO DEL DESARROLLO DEL SOFTWARE

El problema básico del desarrollo del software es el riesgo. Como ejemplos de los riesgos a los que se está expuesto, se puede mencionar:

- Retrasos de planificación: Llega el día de la entrega al cliente pero el software no está disponible.

- Proyecto cancelado: Después de numerosos retrasos, el proyecto se cancela sin haber entrado nunca en producción.

- El sistema se deteriora: El software se pone satisfactoriamente en producción, pero después de un par de años, los costos de hacer cambios o la tasa de defectos crece tanto que el sistema debe ser reemplazado.

- Tasa de defectos: El software se pone en producción, pero la tasa de defectos es tan alta que no se usa.

- Requisitos mal comprendidos: El software se pone en producción, pero no resuelve el requisito planteado inicialmente.

- Cambios en el negocio: El software se pone en producción, pero el problema de negocio para el que se diseñó la solución inicial fue reemplazado por otro problema más acuciante.

- Falsa riqueza de características: El software tiene unas características interesantes, todas las cuales fueron divertidas de programar, pero ninguna de las cuales hace que el cliente gane más dinero.

- Cambios de personal: Después de dos años, todos los buenos programadores del proyecto, comienzan a odiar el programa y se marchan.

XP es una disciplina de desarrollo del software que trata el riesgo en todos los niveles del proceso de desarrollo. XP es también muy productiva, produce software de alta calidad, y es muy divertida de ejecutar.

Extreme programming trata el riesgo comentado anteriormente de la siguiente manera:

- Desviaciones en la planificación: XP propone un ciclo de versión corto, unos pocos meses como máximo, de tal manera que el alcance de cualquier desviación sea limitado. Dentro de una versión, XP realiza iteraciones de una a cuatro semanas sobre las

características requeridas por los clientes para obtener una realimentación lo más refinada posible sobre el proceso. Dentro de una iteración, XP planifica tareas de uno a tres días, para que el equipo pueda resolver problemas en cada iteración.

Finalmente, XP implementa primero las prestaciones de más alta prioridad, de tal manera que cualquier prestación que se pase de fecha tendrá un valor más bajo.

- Proyecto cancelado: XP propone al cliente que elija la versión más pequeña que tenga más sentido para su negocio, así se reduce lo que puede fallar antes de entrar en producción y el valor del software es mayor.

- El sistema se deteriora: XP crea y mantiene con todo detalle un conjunto de pruebas, las cuales se ejecutan y vuelven a ejecutarse después de cada cambio (varias veces al día), para asegurar una calidad mínima.

XP mantiene siempre el sistema en perfecto estado. No se permite que se acumule la suciedad.

- Tasa de defectos: XP realiza pruebas tanto desde la perspectiva de los programadores que escriben las pruebas función a función como de los clientes que escriben pruebas prestación a prestación.

- Requisitos mal comprendidos: XP propone que el cliente se integre como parte del equipo. La especificación del proyecto se refina continuamente durante el desarrollo, de tal manera que lo aprendido por el cliente y el equipo se puede reflejar en el software.

- Cambios en el negocio: XP reduce el ciclo de las versiones, si hay menos cambios durante el desarrollo de una versión. Durante una versión, se favorece que el cliente sustituya nuevas funcionalidades por funcionalidades aun no completadas. El equipo no se da cuenta de si está trabajando sobre una nueva funcionalidad descubierta o sobre otras prestaciones definidas años atrás.

- Falsa riqueza de prestaciones: XP insiste en que solamente se traten las tareas de prioridad más alta.

- Cambios de personal: XP pide a los trabajadores que acepten la responsabilidad de estimar y completar su propio trabajo, les proporciona realimentación sobre el tiempo empleado para que puedan mejorar sus estimaciones, y respeta dichas estimaciones. Las reglas que indican quien puede hacer cambiar las estimaciones son claras. Así, hay menos posibilidad de que el programador se frustre cuando se le pide que haga lo que evidentemente es imposible. XP también anima al contacto humano entre el equipo, reduciendo el aislamiento, que es a menudo el origen de la insatisfacción en el trabajo. Finalmente, XP incorpora un modelo explícito de cambio de personal. Se anima a los

nuevos miembros del equipo a aceptar gradualmente más y más responsabilidad, y son asistidos durante su trabajo por otros y por los trabajadores existentes.

Desde la formación de XP como metodología, los equipos de desarrollo que la han adoptado se han enfrentado con dificultades en su implementación. Esto se debe a que las 12 prácticas de dicha metodología, como puede ser la programación en parejas, el "testing" continuo, la integración continua, etc, no solo tienen su problemáticas individual, sino también la problemática que surge debido a sus interdependencias.

Por tanto, hay que saber cuáles son los problemas a los que los equipos de desarrollo se han enfrentado en sus primeros años de aplicación, para poder ayudar a implantar XP correctamente.

Si aceptamos que el riesgo de un proyecto es el problema a resolver, lo que necesitamos hacer es inventar un estilo de desarrollo de software que trate esos riesgos. Necesitamos comunicar esta disciplina tan claramente como sea posible a los programadores, directores y clientes. Necesitamos disponer de pautas para adaptarla a condiciones locales (es decir, comunicar lo que es fijo y lo que es variable). (Beck, 2000)

5. ECONOMÍA DEL DESARROLLO DEL SOFTWARE

Necesitamos hacer nuestro desarrollo de software económicamente más valioso gastando el dinero más lentamente, obteniendo beneficios más rápidamente, e incrementando la cantidad de tiempo en que el proyecto siga siendo productivo. Pero sobre todo necesitamos aumentar las opciones para las decisiones de negocio.(Beck, 2000)

Sumando el flujo de caja de entrada y salida del proyecto, podemos analizar de forma sencilla qué es lo que hace que un proyecto de software sea valioso. Teniendo en cuenta el efecto del tipo de interés, podemos calcular el valor neto actual del flujo de caja. Podemos además refinar nuestro análisis, multiplicando los flujos de caja descontados por la probabilidad que el proyecto tenga de sobrevivir para pagar o ganar esos flujos de caja.

Con estos tres factores:

- Flujos de caja de entrada y salida.
- Tipos de interés
- Mortalidad del proyecto

podemos crear una estrategia para maximizar el valor económico del proyecto. Podemos hacer esto de la manera siguiente:

Gastando menos, lo cual es difícil porque cada uno comienza casi con las mismas habilidades y herramientas.

Gastando más, lo cual es solamente posible con una organización excelente de marketing y ventas.

Gastando más tarde y ganando antes, así pagaremos menos intereses del dinero que gastemos y ganaremos más intereses del dinero que recibamos.

Aumentando la probabilidad de que el proyecto siga vivo, de tal forma que tengamos más probabilidad de obtener el pago importante de finalización del proyecto.

Hay otra forma de abordar la economía de un proyecto software, y es observándolo como una sucesión de opciones. En la gestión de un proyecto software pueden observarse cuatro clases de opciones:

• Opción de abandonar: Se puede obtener algo del proyecto incluso si se cancela. Cuanto más valor se pueda obtener del proyecto sin entregarlo realmente en su forma ideada inicialmente, mejor.

• Opción de cambiar: Se puede cambiar la dirección del proyecto. Una estrategia de gestión de proyectos es más valiosa si a mitad del proyecto los clientes pueden cambiar los requisitos. Cuanto mayor sea la frecuencia y profundidad de los requisitos que se pueda hacer, mejor.

• Opción de aplazar: Se puede esperar hasta que la situación se resuelva por sí misma antes de invertir. Una estrategia de gestión de proyectos es más valiosa si se puede esperar a gastar el dinero sin perder totalmente la posibilidad de inversión. Cuanto mayor sea el aplazamiento y mayor la cantidad de dinero que se pueda aplazar, mejor.

• Opción de crecer: Si un mercado empieza a tener éxito, se puede crecer rápidamente para obtener ventaja de ello. Una estrategia de gestión de proyectos es más valiosa si puede crecer la producción proporcionalmente al crecimiento de la inversión. Cuanto más rápido y mayor sea el crecimiento que un proyecto pueda tener, mejor.

Hay cinco factores implicados:

- La cantidad de inversión requerida para obtener la opción.
- El precio al que se pueda comprar otra cosa si se procede con la opción.
- El valor actual de la otra cosa.
- La cantidad de tiempo en que se puede ejercer las opciones.
- La incertidumbre en el valor final del precio.

De todas ellas, el valor de las opciones está generalmente determinado por el último factor, la incertidumbre. A partir de esto podemos hacer una predicción concreta. Supongamos que creamos una estrategia de gestión de proyectos que maximice el valor del proyecto analizado mediante opciones que promocionan:

- Realimentación precisa y frecuente sobre el progreso.
- Muchas oportunidades de cambiar drásticamente los requisitos.
- Una menor inversión inicial.
- La oportunidad de avanzar rápidamente.

Cuanto mayor es la incertidumbre, más valiosa será la estrategia. Esto es cierto si la incertidumbre proviene del riesgo técnico, de un entorno de negocios cambiante, o de una evolución rápida de los requisitos. (Beck, 2000)

6. METODOLOGÍA EXTREME PROGRAMMING

La metodología XP define cuatro variables para cualquier proyecto de software:

- Coste
- Tiempo
- Calidad
- Ámbito

La forma de hacer en este modelo el juego del desarrollo del software, es que las fuerzas externas (clientes, directores de proyecto) eligen los valores de tres variables cualesquiera. El equipo de desarrollo elige el valor resultante de la cuarta variable.

Se deben hacer visibles las cuatro variables. Si cualquiera, programadores, clientes y directores de proyecto, pueden ver las cuatro variables, pueden conscientemente escoger las variables a controlar. Si no les gusta el resultado que corresponde a la cuarta variable, pueden cambiar las entradas, o pueden elegir otras tres variables diferentes a controlar. (Beck, 2000)

Las interacciones entre las variables son las siguientes:

• Coste: Mucho dinero puede engrasar la maquinaria un poco, pero demasiado dinero pronto crea más problemas que resuelve. Por otra parte, si damos a un proyecto muy poco dinero, no seremos capaces de resolver el problema del negocio del cliente.

• Tiempo: Disponer de más tiempo para la entrega puede mejorar la calidad e incrementar el ámbito. Ya que la realimentación desde los sistemas en producción es de mayor calidad que cualquier otra clase de realimentación, dar a un proyecto demasiado tiempo será perjudicial. Si damos a un proyecto poco tiempo, la calidad sufre, con el ámbito, tiempo y coste no muy atrás.

• Calidad: La calidad es terrible como variable de control. Puedes obtener beneficios a corto plazo (días o semanas) sacrificando deliberadamente la calidad, pero el coste, humano, de negocio y técnico, es enorme.

• Ámbito: Menos ámbito hace posible entregar mejor calidad (mientras el problema del cliente esté todavía resuelto). También permite entregar más rápido y barato.

No hay una relación sencilla entre las cuatro variables. Por ejemplo, no se puede obtener software más rápido, gastando más dinero.

En muchos sentidos, el coste es la variable más limitada. No se puede gastar solo en calidad, o en ámbito, o ciclos de entrega cortos. De hecho, al comienzo de un proyecto, no se puede gastar mucho. La inversión tiene que comenzar siendo pequeña y crecer con el tiempo. Después de un tiempo, se puede, de forma productiva, gastar más y más dinero.

Todas las restricciones sobre el coste pueden volver locos a los directores de proyecto. Especialmente si están sujetos a un proceso de presupuesto anual, ellos están tan acostumbrados a considerarlo todo desde la perspectiva del coste que cometerán grandes errores al ignorar las restricciones sobre cuánto control proporciona el coste.

El otro problema con el coste es que mayores costes a menudo alimentan objetivos tangenciales, como estatus o prestigio.

Por otra parte, el coste está profundamente relacionado con las otras variables. Dentro del rango de inversión que pueda sensatamente hacerse, gastando más dinero se puede aumentar el ámbito, o se puede intentar de forma más deliberada aumentar la calidad, o puedes (hasta cierto punto) reducir el tiempo de salida al mercado.

Gastando dinero, también se puede reducir las desavenencias: máquinas más rápidas, más especialistas técnicos, mejores oficinas. (Beck, 2000)

Las restricciones de controlar proyectos controlando el tiempo, generalmente vienen de fuera. Así, la variable tiempo está a menudo fuera de las manos del director del proyecto y está en manos del cliente.

La Calidad es otra variable extraña. A menudo, el insistir en la mejora de la calidad puede hacer que el proyecto esté listo antes, o puede conseguir hacer más en una cantidad de tiempo dada.

Hay una extraña relación entre la calidad interna y la externa. La calidad externa es la calidad que mide el cliente. La calidad interna es la calidad que miden los programadores. Sacrificar temporalmente la calidad interna para reducir el tiempo de salida al mercado del producto, con la esperanza de que la calidad externa no se vea muy dañada es tentador a corto plazo. Y se puede con frecuencia hacer impunemente generando una confusión en cuestión de semanas o meses. Al fin y al cabo, los problemas de calidad interna hacen que el software sea prohibitivamente caro de mantener, o incapaz de alcanzar un nivel de competitividad de calidad externa.

Hay un efecto humano en la calidad. Cualquiera quiere hacer un buen trabajo, y se trabaja mejor si se siente que se está haciendo un buen trabajo. Si deliberadamente se degrada la calidad, el equipo puede ir más rápido al principio, pero pronto la desmoralización de producir un software basura aplastará cualquier ganancia que temporalmente se haya obtenido de no hacer las pruebas, o las revisiones, o utilizar estándares.

Muchas personas saben que el coste, la calidad y el tiempo son variables de control, pero no conocen la cuarta variable. Para el desarrollo del software, el ámbito es la variable más importante a tener en cuenta. Los programadores y el personal de negocio no tienen más que una idea vaga sobre lo que tiene valor en el software que se está desarrollando. Una de las decisiones más importantes en la gestión del proyecto es la eliminación del ámbito. Si se gestiona activamente el ámbito, se puede proporcionar a los directores de proyecto y clientes de control sobre el coste, calidad y tiempo.

Si creamos una disciplina de desarrollo basada en este modelo, podríamos fijar la fecha, calidad y coste de un software. Podríamos observar el ámbito implicado por las tres primeras variables. Entonces, conforme progresa el desarrollo, podríamos continuamente ajustar el ámbito para adaptarlo a las condiciones conforme las vamos encontrando.

Debería ser un proceso que tolerase el cambio fácilmente, porque el proyecto podría cambiar de rumbo con frecuencia. Nadie desea gastar tiempo en desarrollar un software que después no fuese utilizado. Nadie desea construir un camino que nunca se utiliza porque se coge otro. También, tendríamos que tener un proceso que hiciese que el coste de los cambios fuese razonable a lo largo de la vida del sistema.

Si se deja fuera importante funcionalidad al final de cada ciclo de versión, el cliente quedará disgustado. Para evitar esto, XP utiliza dos estrategias:

- Consigue mucha práctica haciendo estimaciones y realimentado los resultados reales. Mejores estimaciones reducen la probabilidad de que se tenga que dejar fuera funcionalidad.

- Implementa en primer lugar los requisitos más importantes del cliente, de manera que si se deja después alguna funcionalidad, es menos importante que la funcionalidad que ya está incorporada al sistema.

Los ciclos de vida “tradicionales” proponen una clara distinción entre las etapas del proyecto de software, y tienen un plan bien preestablecido acerca del proceso de desarrollo. Asimismo, en todos ellos se parte de especificaciones claras, si no del total del proyecto, por lo menos de una buena parte inicial.(Joskowicz, 2008)

El ciclo de vida de un proyecto XP incluye, al igual que las otras metodologías, entender lo que el cliente necesita, estimar el esfuerzo, crear la solución y entregar el producto final al cliente. Sin embargo, XP propone un ciclo de vida dinámico, donde se admite expresamente que, en muchos casos, los clientes no son capaces de especificar sus requerimientos al comienzo de un proyecto.

Típicamente un proyecto XP lleva 10 a 15 ciclos o iteraciones.

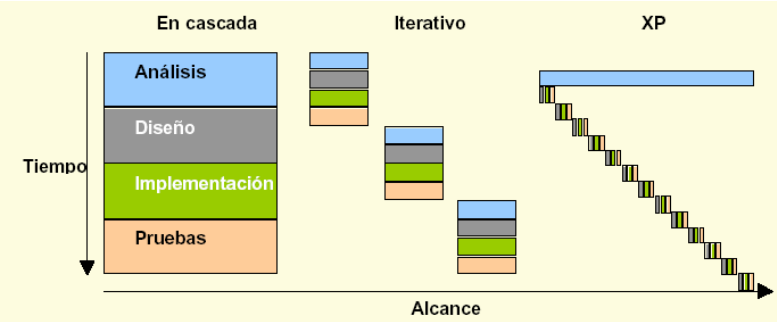


Figura 3: Comparación ciclos de vida tradicionales con el de XP

El ciclo de vida de XP se enfatiza en el carácter interactivo e incremental del desarrollo, Según una iteración de desarrollo es un período de tiempo en el que se realiza un conjunto de funcionalidades determinadas que en el caso de XP corresponden a un conjunto de historias de usuarios. (Anaya, A.)

Las iteraciones son relativamente cortas ya que se piensa que cuanto más rápido se le entreguen desarrollos al cliente, más retroalimentación se va a obtener y esto va a representar una mejor calidad del producto a largo plazo. Existe una fase de análisis inicial orientada a programar las iteraciones de desarrollo y cada iteración incluye diseño, codificación y pruebas, fases superpuestas de tal manera que no se separen en el tiempo.

La siguiente figura muestra las fases en las que se subdivide el ciclo de vida XP:

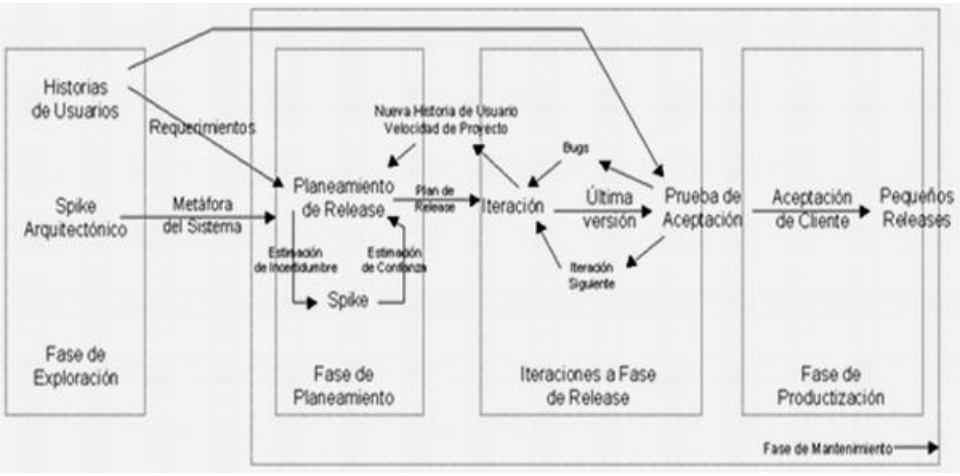


Figura 4: Ciclo de vida Extreme Programming

El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

- El cliente define el valor de negocio a implementar.
- El programador estima el esfuerzo necesario para su implementación.
- El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
- El programador construye ese valor de negocio.
- Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

El ciclo de vida ideal de XP consiste de siete fases: Exploración, Planificación de la Entrega (Release), Diseño, Producción, Pruebas, Mantenimiento y Muerte del Proyecto.

Un proyecto XP comienza con un periodo de exploración. El objetivo de la exploración es identificar, priorizar y estimar los requisitos. Una vez que se hayan identificado requisitos suficientes como para proporcionar el sistema más pequeño posible que aporte valor al cliente, se planifica la primera versión. Conforme transcurre el tiempo, otras exploraciones proporcionarán más requisitos y se planificarán otras versiones. (Newkirk y Martin, 2001)

Las versiones se dividen en varias iteraciones. Se escribe software en cada iteración, y cada iteración entrega algo de valor al cliente.

6.1 Fase de Exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.

Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

Durante la exploración, o en cualquier momento que el equipo necesite resolver incertidumbres o mitigar el riesgo, ellos pueden preparar un punto de fijación. Un punto de fijación es una prueba muy rápida que llega a profundizar sobre el aspecto en cuestión.(Joskowicz, J., 2008)

6.1.1 Historias de Usuario

La exploración da lugar a un documento de requisitos. Los programadores y el cliente se reúnen y discuten las necesidades del cliente. El cliente escribe historias que describen estas necesidades. En la discusión con el cliente, los programadores eliminan la ambigüedad de las historias y aseguran que éstas puedan ser estimadas y probadas. Los clientes se aseguran de que las historias sean significativas en términos de valor para su negocio.
Las historias se escriben normalmente en tarjetas. No contienen mucho texto. Más bien, sirven como recordatorio de las conversaciones mantenidas entre los programadores y el cliente. Más adelante, durante cada iteración, el cliente proporciona un detalle escrito sobre las historias por medio de pruebas de aceptación.

Una historia debe ser lo suficientemente pequeña como para que el equipo la desarrolle durante una iteración (una a tres semanas). Las historias pequeñas son mejores que las grandes. Una historia debe ser probada. El cliente deberá ser capaz de especificar las pruebas de aceptación que verifiquen que la historia es correcta y completa. Superar estas pruebas es la definición de haber concluido una historia. Finalmente, el cliente debe ser capaz de priorizar cada historia. Una historia debería tener una única responsabilidad que condujese a una única prioridad. Si se parte de que una historia es más importante que otras, la historia se debería dividir en dos o más historias pequeñas. .(Joskowicz, J., 2008)

La estimación de las historias se escriben en las tarjetas y las hacen únicamente los programadores. Son expresadas en unidades arbitrarias de esfuerzo y son proporcionales a la cantidad de tiempo que llevará la tarea. Algunos programadores utilizan la semana ideal de programación.

Una semana ideal de programación es la cantidad de trabajo que puedes realizar durante una semana sin llamadas telefónicas, sin reuniones, sin tener que trabajar con nadie, en la que nadie moleste, es decir, sin distracciones.

Otro de los estimadores que los programadores utilizan son los llamados Puntos de Historia.

Es el único mecanismo que determina el tamaño de una historia de usuario, es decir, es la misma medida para cualquier miembro del grupo. En otras palabras, si se estima una historia en 5 puntos de historia, aunque lo desarrolle un desarrollador junior o un desarrollador sénior, siguen siendo 5 puntos de historias.

Los puntos de historia son el tamaño relativo de una historia comparada con otras historias estimadas por el mismo equipo. No importa quién implemente la historia, ni cuando le vaya a tomar. Si, por el contrario, se estima en horas o días, puede ser una tarea de 2 días para un desarrollador sénior y de 1 semana para uno junior.

Una vez que el equipo gane experiencia estimado en puntos de historia, la estimación se hace más precisa y puede ser realizada muy rápidamente en comparación con las técnicas tradicionales de estimación.

Los Puntos de Historia sirven para valorar el "esfuerzo" necesario para desarrollar cada "Historia de Usuario". Los valores siguen una sucesión de Fibonacci aproximada, y redondeada. No nos interesa afinar demasiado, sobre todo en los valores más altos, ya que son los más susceptibles de desviarse. Los Puntos de Historia sirven para comparar Historias. Es decir, si dos Historias están estimadas en "5", costará el mismo esfuerzo (aproximadamente) llevarlas a cabo, y seguramente se tarde lo mismo en ser concluidas. Si una Historia está estimada con valores altos, 100 o infinito, seguramente deberá ser dividida en varias Historias "más pequeñas". El infinito se suele usar cuando una Historia no está suficientemente definida, y normalmente conlleva una rescritura de la Historia por parte del cliente o el Product Owner.

El Equipo debe discutir y debatir cada uno de las estimaciones, y llegar a un acuerdo sobre los Puntos de Historia de una Historia de Usuario.

Historia de Usuario	
Número:	**Nombre Historia de Usuario:**
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre):	
Usuario:	**Iteración Asignada:**
Prioridad en Negocio: (Alta / Media / Baja)	**Puntos Estimados:**
Riesgo en Desarrollo: (Alto / Medio / Bajo)	**Puntos Reales:**
Descripción:	
Observaciones:	

Figura 5: Historia de usuario

6.2 Fase de Planificación

Un proyecto XP se divide en una serie de versiones. Cada versión proporciona un valor de negocio al cliente. Cuando se planifica una versión el cliente selecciona las historias que serán implementadas. Esta selección puede que no sea la más eficiente desde el punto de

vista técnico, pero asegura que cada versión proporciona el máximo valor al negocio. El valor de negocio deja a un lado la eficiencia técnica.

Una versión normalmente lleva de uno a tres meses. Cuanto más corta es la versión, más rápidamente se conseguirá realimentación. Sin embargo, una versión también debe proporcionar valor al negocio, y el negocio debe ser capaz de absorber el valor. Al final el cliente decide cómo de larga será la versión.

Cuando se planifica cada versión, hay que seguir una regla sencilla. El equipo no puede comprometerse a hacer más trabajo que el que había hecho en la versión anterior. Este trabajo se mide en términos de estimación que se escriben en las tarjetas. La suma de estos números en una versión se conoce como la velocidad de la versión. Para la primera versión el equipo elige una velocidad razonable pero arbitraria, pues no se basa en medidas reales.

Muchos proyectos fijan la duración de la versión y eligen historias para que encajen. Otros proyectos pueden escoger varias historias y ajustar la duración; sin embargo, esto lleva a menudo a versiones más largas, que pueden ser difíciles de gestionar. (Newkirk y Martin, 2001)

Esta técnica delimita la responsabilidad entre el cliente y los programadores. El cliente decide el contenido. Los programadores proporcionan las estimaciones. La velocidad sencillamente es la cantidad de trabajo realizado en versiones anteriores. No se permite que el cliente influya en las estimaciones, ni a los programadores cambiar el contenido, y a nadie hacer segundas conjeturas de la velocidad.

Una versión se descompone en iteraciones de una a tres semanas de duración. La longitud de la iteración se elige al principio del proyecto y permanece constante después.

El equipo comienza planificando la iteración dándole al cliente una estimación. Esta estimación es la cantidad de trabajo que los desarrolladores piensan que pueden conseguir hacer. Esta cantidad se expresa en las mismas unidades que las estimaciones de las historias.

Se debe aplicar una sencilla regla: La cantidad de trabajo que los desarrolladores pueden comprometer en una iteración es la misma que pudieron hacer en la última iteración. Para la primera iteración se proporciona una estimación arbitraria razonable. Será errónea, pero en las siguientes iteraciones se corregirá rápidamente. .(Joskowicz, J., 2008)

El cliente elige las historias que se implementarán completamente en la iteración y acepta no cambiar o añadir nada hasta que la iteración se complete. El cliente también especifica las pruebas de aceptación que indican la terminación satisfactoria de las historias.

Los programadores dividen las historias en tareas y determinan su orden de implementación. La duración de las tareas no debe ser superior a uno o dos días.

Las tareas no son asignadas, los programadores se comprometen con ellas en las que quieren trabajar. Ninguna tarea se puede dejar sin elegir.

Los programadores estiman las tareas que eligen considerando aquellas que ellos han concluido en la última iteración. No se permite que los programadores se responsabilicen de más tareas de las que han completado en la última iteración, por lo que puede ser necesaria una nivelación.

Las estimaciones se suman y se comparan con la duración de la iteración. Si hay mucho más trabajo a realizar, el cliente decide que trasladará de esta iteración a la siguiente. El cliente puede trasladar historias enteras o parciales a la futura iteración. Si las estimaciones no completan la iteración, el cliente suministrará más historias.

De nuevo hay una delimitación de responsabilidad. Los clientes eligen el contenido (historias) y especifican los criterios de terminación (pruebas de aceptación). Los programadores proporcionan las estimaciones y eligen la estrategia de implementación.

6.2.1 Plan de entregas ("Release Plan")

El cronograma de entregas establece qué historias de usuario serán agrupadas para conformar una entrega, y el orden de las mismas. Este cronograma será el resultado de una reunión entre todos los actores del proyecto (cliente, desarrolladores, gerentes, etc.). XP denomina a esta reunión "Juego de planificación" ("Planning game"), pero puede denominarse de la manera que sea más apropiada al tipo de empresa y cliente (por ejemplo, Reunión de planificación, "Planning meeting" o "Planning workshop").
Típicamente el cliente ordenará y agrupará según sus prioridades las historias de usuario. El cronograma de entregas se realiza en base a las estimaciones de tiempos de desarrollo realizadas por los desarrolladores.

Luego de algunas iteraciones es recomendable realizar nuevamente una reunión con los actores del proyecto, para evaluar nuevamente el plan de entregas y ajustarlo si es necesario.

6.2.2 Plan de iteraciones ("Iteration Plan")

Las historias de usuarios seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo al orden preestablecido.
Al comienzo de cada ciclo, se realiza una reunión de planificación de la iteración.

Cada historia de usuario se traduce en tareas específicas de programación. Asimismo, para cada historia de usuario se establecen las pruebas de aceptación. Estas pruebas se realizan al final del ciclo en el que se desarrollan, pero también al final de cada uno de los ciclos siguientes, para verificar que subsiguientes iteraciones no han afectado a las anteriores.

Las pruebas de aceptación que hayan fallado en el ciclo anterior son analizadas para evaluar su corrección, así como para prever que no vuelvan a ocurrir.

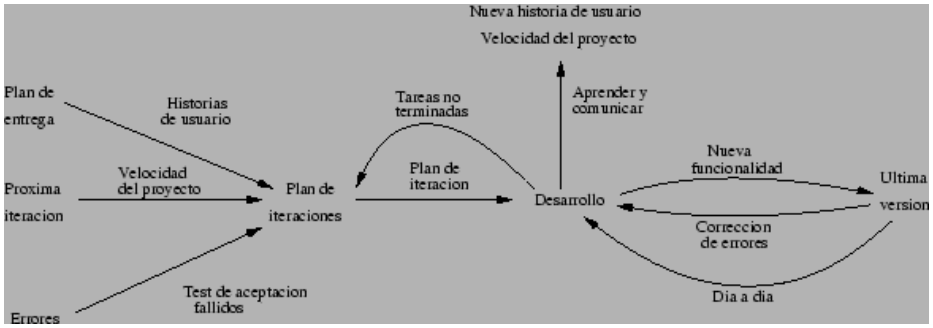


Figura 6: Panificacion de iteraciones

6.2.3 Reuniones diarias de seguimiento (“Stand-up meeting”)

El objetivo de tener reuniones diarias es mantener la comunicación entre el equipo, y compartir problemas y soluciones. En la mayoría de estas reuniones, gran parte de los participantes simplemente escuchan, sin tener mucho que aportar. Para no quitar tiempo innecesario del equipo, se sugiere realizar estas reuniones en círculo y de pie.

6.3 Fase de Diseño

La metodología XP hace especial énfasis en los diseños simples y claros.

Un diseño simple se implementa más rápidamente que uno complejo. Por ello XP propone implementar el diseño más simple posible que funcione. Se sugiere nunca adelantar la implementación de funcionalidades que no correspondan a la iteración en la que se esté trabajando. .(Joskowicz, J., 2008)

Cuando aparecen problemas técnicos, o cuando es difícil de estimar el tiempo para implementar una historia de usuario, pueden utilizarse pequeños programas de prueba (llamados “spike”), para explorar diferentes soluciones. Estos programas son únicamente para probar o evaluar una solución, y suelen ser desechados luego de su evaluación.

6.3.1 Recodificación

La recodificación (“refactoring”) consiste en escribir nuevamente parte del código de un programa, sin cambiar su funcionalidad, a los efectos de hacerlo más simple, conciso y/o entendible. Muchas veces, al terminar de escribir un código de programa, se piensa que, si se comienza de nuevo, se hubiera hecho en forma diferente, más clara y eficientemente. Sin

embargo, como ya está pronto y "funciona", rara vez es reescrito. Las metodologías de XP sugieren recodificar cada vez que sea necesario. Si bien, puede parecer una pérdida de tiempo innecesaria en el plazo inmediato, los resultados de ésta práctica tienen sus frutos en las siguientes iteraciones, cuando sea necesario ampliar o cambiar la funcionalidad. La filosofía que se persigue es, como ya se mencionó, tratar de mantener el código más simple posible que implemente la funcionalidad deseada.

6.4 Fase de Producción

Requiere prueba y comprobación extra del funcionamiento del sistema antes de que éste se pueda liberar al cliente. En esta fase, los nuevos cambios pueden todavía ser encontrados y debe tomarse la decisión de si se incluyen o no en el release actual. Durante esta fase, las iteraciones pueden ser aceleradas de una a tres semanas. Las ideas y las sugerencias pospuestas se documentan para una puesta en práctica posterior, por ejemplo en la fase de mantenimiento. Después de que se realice el primer release productivo para uso del cliente, el proyecto de XP debe mantener el funcionamiento del sistema mientras que realiza nuevas iteraciones.

Aunque los programadores hayan tomado a nivel individual la responsabilidad de las tareas, todo el software de producción se escribe por parejas de programadores. Cada pareja trabaja en una estación de trabajo compartiendo el teclado.

6.4.1 Programación en Pareja

XP propone que se desarrolle en parejas de programadores, ambos trabajando juntos en un mismo ordenador. Si bien parece que ésta práctica duplica el tiempo asignado al proyecto (y por ende, los costos en recursos humanos), al trabajar en pareja se minimizan los errores y se logran mejores diseños, compensando la inversión en horas. El producto obtenido es por lo general de mejor calidad que cuando el desarrollo se realiza por programadores individuales.

En un estudio realizado por Cockburn y Williams, se concluye que la programación en pareja tiene un sobre costo aproximado de 15%, y no de un 100% como se puede pensar a priori. Este sobre costo es rápidamente pagado por la mejor calidad obtenida en el producto final.

Adicionalmente, la programación en pareja tiene las siguientes ventajas:

- La mayoría de los errores se descubren en el momento en que se codifican, ya que el código es permanentemente revisado por dos personas.

- La cantidad de defectos encontrados en las pruebas es estadísticamente menor.

- Los diseños son mejores y el código más corto.

• El equipo resuelve problemas en forma más rápida.

• Las personas aprenden significativamente más, acerca del sistema y acerca de desarrollo de software.

• El proyecto termina con más personas que conocen los detallas de cada parte del código.

• Las personas aprenden a trabajar juntas, generando mejor dinámica de grupo y haciendo que la información fluya rápidamente.

• Las personas disfrutan más de su trabajo.

Uno de los requerimientos de XP es tener al cliente disponible durante todo el proyecto. No solamente como apoyo a los desarrolladores, sino formando parte del grupo. El involucramiento del cliente es fundamental para que pueda desarrollarse un proyecto con la metodología XP.

Al comienzo del proyecto, el cliente debe proporcionar las historias de usuarios. Pero, dado que estas historias son expresamente cortas y de "alto nivel", no contienen los detalles necesarios para realizar el desarrollo del código. Estos detalles deben ser proporcionados por el cliente, y discutidos con los desarrolladores, durante la etapa de desarrollo. No se requieren de largos documentos de especificaciones, sino que los detalles son proporcionados por el cliente, en el momento adecuado, "cara a cara" a los desarrolladores.

Si bien esto parece demandar del cliente recursos por un tiempo prolongado, debe tenerse en cuenta que en otras metodologías este tiempo es insumido por el cliente en realizar los documentos detallados de especificación.

Adicionalmente, al estar el cliente en todo el proceso, puede prevenir a tiempo de situaciones no deseables, o de funcionamientos que no eran los que en realidad se deseaban. En otras metodologías, estas situaciones son detectadas en forma muy tardía del ciclo de desarrollo, y su corrección puede llegar a ser muy complicada.

6.4.2 Uso de estándares

XP promueve la programación basada en estándares, de manera que sea fácilmente entendible por todo el equipo, y que facilite la recodificación.

6.4.3 Integraciones permanentes

Todos los desarrolladores necesitan trabajar siempre con la "última versión".

Realizar cambios o mejoras sobre versiones antiguas causan graves problemas, y retrasa al proyecto. Es por eso que XP promueve publicar lo antes posible las nuevas versiones,

aunque no sean las últimas, siempre que estén libres de errores. Idealmente, todos los días deben existir nuevas versiones publicadas.

Para evitar errores, solo una pareja de desarrolladores puede integrar su código a la vez.

6.4.4 Propiedad colectiva del código

En un proyecto XP, todo el equipo puede contribuir con nuevas ideas que apliquen a cualquier parte del proyecto. Asimismo, cualquier pareja de programadores puede cambiar el código que sea necesario para corregir problemas, agregar funciones o recodificar.

En otras metodologías, este concepto puede parecer extraño. Muchas veces se asume que, si hay algo de propiedad colectiva, la responsabilidad también es colectiva. Y que "todos sean responsables", muchas veces significa que "nadie es responsable". En este caso, quienes encuentran un problema, o necesitan desarrollar una nueva función, pueden resolverlo directamente, sin necesidad de "negociar" con el "dueño" o autor del módulo (ya que, de hecho, este concepto no existe en XP).

Muchas veces, explica Cunningham, una solución pasa por la recodificación de varios módulos, que atraviesan de forma horizontal una determinada jerarquía vertical. Si es necesario dialogar y convencer al encargado de cada módulo, posiblemente la solución no se pueda implementar, por lo menos en tiempos razonables. En XP, se promueve la recodificación, en tareas de generar códigos más simples y adaptados a las realidades cambiantes. Cualquier pareja de programadores puede tomar la responsabilidad de este cambio. Los testeos permanentes deberían de aseguran que los cambios realizados cumplen con lo requerido, y además, no afectan al resto de las funcionalidades.

6.4.5 Ritmo sostenido

La metodología XP indica que debe llevarse un ritmo sostenido de trabajo.

Anteriormente, ésta práctica se denominaba "Semana de 40 horas". Sin embargo, lo importante no es si se trabajan, 35, 40 o 42 horas por semana. El concepto que se desea establecer con esta práctica es el de planificar el trabajo de manera de mantener un ritmo constante y razonable, sin sobrecargar al equipo.

Cuando un proyecto se retrasa, trabajar tiempo extra puede ser más perjudicial que beneficioso. El trabajo extra desmotiva inmediatamente al grupo e impacta en la calidad del producto. En la medida de lo posible, se debería renegociar el plan de entregas ("Release Plan"), realizando una nueva reunión de planificación con el cliente, los desarrolladores y los gerentes. Adicionalmente, agregar más desarrolladores en proyectos ya avanzados no siempre resuelve el problema.

6.5 Fase de Pruebas:

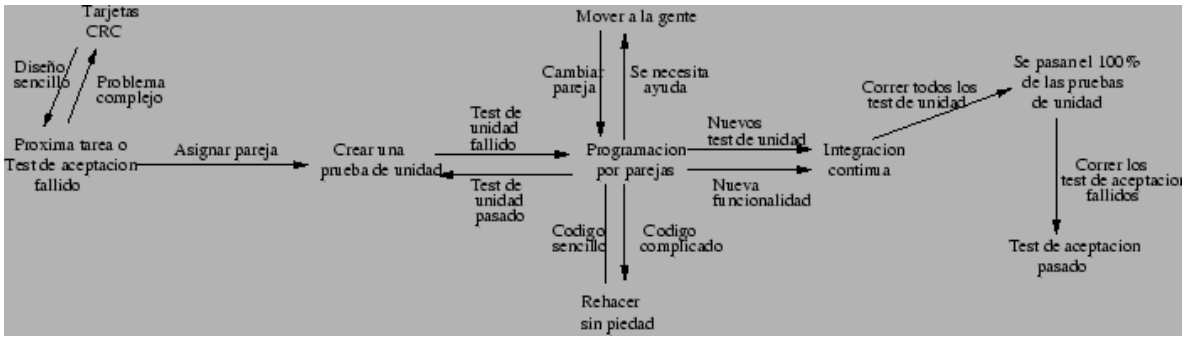


Figura 7: Fase de Pruebas

6.5.1 Pruebas unitarias

Lo primero que se hace antes de programar un módulo es preparar la prueba de unidad. Esto indica al programador que es lo que tiene que hacer cuando codifica. Los requisitos aparecen en forma de pruebas de unidad. Se sabe cuándo se ha terminado porque se superan todos los tests de unidad. El beneficio que tiene de ello el diseño, es que en los tests de unidad se pone aquello que es importante para el cliente. Una forma de hacer esto es a base de pequeños incrementos: (Álvarez, J.R. y Arias, M.)

- Se crea una prueba de unidad pequeña que refleja parte de los requisitos.
- Se hace el código que satisface ese test.
- Se crea una segunda prueba.
- Se añade el código correspondiente.

Veamos ahora como se hace una prueba de unidad. Hay tres pasos a seguir:

- Se crea un armazón o patrón para poder hacer partiendo de él las pruebas de unidad de un modo automatizado.
- Se hace un test de todas las clases del sistema.
- Se debe escribir el test antes de codificar el módulo.

La idea de hacer el test antes del código es importante, porque si se deja para el final, es posible que se encuentren problemas inesperados y se necesite más tiempo del inicialmente previsto. Además, en realidad el test no se hace antes sino durante porque se construye de forma incremental, pero siempre el test antes del código. (Álvarez, J.R. y Arias, M.)

Cuando se publica un módulo al almacén debe ir obligatoriamente acompañado del test correspondiente, y no se puede publicar código que no haya pasado todos los tests. Como el código no tiene un propietario fijo y todo el mundo puede hacer modificaciones, esta norma es bastante razonable, cualquiera puede modificar una línea de código que haya escrito otra persona, ahora bien, la modificación tiene que pasar todos los tests asociados a la unidad.

Los tests de unidad permiten también rehacer el código porque pueden comprobar si un cambio en la estructura supone un cambio en la funcionalidad. Un pequeño problema es que los tests en si mismos pueden tener errores. Por otra parte, una de las cosas deseables es poder hacer frecuentemente integraciones de todo el código. Si se construye un conjunto de tests globales para comprobar el producto final, será posible comprobar rápidamente si los cambios hechos en una unidad se integran bien y de esta forma no dejarlo todo para el final.

6.5.2 Programación dirigida por las pruebas ("Test-driven programming")

En las metodologías tradicionales, la fase de pruebas, incluyendo la definición de los tests, es usualmente realizada sobre el final del proyecto, o sobre el final del desarrollo de cada módulo. La metodología XP propone un modelo inverso, en el que, lo primero que se escribe son los test que el sistema debe pasar. Luego, el desarrollo debe ser el mínimo necesario para pasar las pruebas previamente definidas.

Las pruebas a las que se refiere ésta práctica, son las pruebas unitarias, realizadas por los desarrolladores. La definición de estos test al comienzo, condiciona o "dirige" el desarrollo.

6.5.3 Detección y corrección de errores

Cuando se encuentra un error ("bug"), éste debe ser corregido inmediatamente, y se deben tener precauciones para que errores similares no vuelvan a ocurrir.

Asimismo, se generan nuevas pruebas para verificar que el error haya sido resuelto.

6.5.4 Pruebas de aceptación

El test de aceptación también se conoce como test funcional en otros sitios. Los tests de aceptación son cajas negras que comprueban el funcionamiento del sistema. Son escritos por el cliente, y es el cliente el responsable de que sea correcto. También es el cliente el que decide la prioridad de cada test fallido. También se usan como test de regresión. Para que se pueda comprobar con rapidez y muchas veces si las historias de usuario cumplen con los tests de aceptación, estos deberían estar automatizados. Los resultados se comunican al grupo, que debe gestionar el tiempo para corregir errores. .(Joskowicz, J., 2008)

A partir de las historias de usuario se hacen los tests de aceptación y a cada una le puede corresponder uno o varios. No se considera que una historia ha sido completada hasta que no pase todos sus tests de aceptación. Al igual que con los tests de unidad, los tests de aceptación deben hacerse antes de empezar a depurar. La diferencia entre un test de aceptación y un test de unidad está clara: un test de aceptación comprueba los requisitos expresados en las historias de usuario y un test de unidad comprueba que el código de una unidad es correcto. (Álvarez, J.R. y Arias, M.)

Caso de Prueba de Aceptación	
Código:	**Historia de Usuario (Nro. y Nombre):**
Nombre:	
Descripción:	
Condiciones de Ejecución:	
Entrada / Pasos de ejecución:	
Resultado Esperado:	
Evaluación de la Prueba:	

Figura 8: Prueba de aceptación

6.6 Fase de mantenimiento:

Requiere de un mayor esfuerzo para satisfacer también las tareas del cliente. Así, la velocidad del desarrollo puede desacelerar después de que el sistema esté en la producción. La fase de mantenimiento puede requerir la incorporación de nueva gente y cambiar la estructura del equipo. (Joskowicz, J., 2008)

6.7 Fase de muerte:

Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo. .(Joskowicz, J., 2008)

7. VALORES DE EXTREME PROGRAMMING

XP se basa en cuatro valores, que deben estar presentes en el equipo de desarrollo para que el proyecto tenga éxito.

- Comunicación
- Sencillez
- Realimentación
- Valentía

7.1 Comunicación

El primero valor de XP es la comunicación. Los problemas en los proyectos se pueden remontar a alguien que no dijo a otro algo importante. Algunas veces, un programador no le dice a otro algún cambio crítico en el diseño. En otras ocasiones, un programador no le hace al cliente alguna pregunta adecuada, de tal forma que una decisión importante no se considera. Otras veces un director de proyecto no hace al programador la pregunta adecuada, y se informa mal del progreso del proyecto.

La mala comunicación no sucede por casualidad. Hay muchas circunstancias que conducen a una ruptura en la comunicación.

XP ayuda a mantener el flujo de comunicaciones apropiado empleando muchas prácticas que no se pueden hacer sin comunicación. Éstas son prácticas que tienen sentido a corto plazo, como las pruebas, la programación en pareja y las tareas de estimación. El efecto de probar, emparejarse y estimar hace que los programadores, clientes y directores de proyecto tengan que comunicarse.

Esto no significa que la comunicación nunca se atasque en un proyecto XP. Las personas pueden asustarse, cometer errores, distraerse. XP emplea a un preparador cuyo trabajo es observar cuándo la gente no se está comunicando y los vuelve a introducir.(Beck, 2000)

7.2 Sencillez

El segundo valor de XP es la sencillez. El preparador XP le hace la siguiente pregunta a los equipos: ¿Qué es lo más simple que podría funcionar?.

La sencillez no es fácil. Una de las cosas más difíciles del mundo es no mirar lo que necesitarás implementar mañana, la próxima semana y el próximo mes.

XP es apostar. Apuesta que es mucho mejor hacer una cosa sencilla hoy y pagar un poco más mañana para cambiar, si es que es necesario, que hacer una cosa más complicada hoy y no utilizarla después.

La sencillez y la comunicación se apoyan mutuamente. A mayor comunicación, se tiene más claro lo que exactamente se necesita hacer y aumenta la confianza en lo que realmente no necesitas hacer. Cuanto más simple es tu sistema, menos se tiene que comunicar del mismo, lo cual conduce a una comunicación más completa, especialmente si se puede simplificar el sistema lo suficiente como para necesitar pocos programadores.(Beck, 2000)

7.3 Realimentación

El tercer valor en XP es la realimentación. Otras frases del preparador son: "No me preguntes a mí, pregúntale al sistema", y "¿Has escrito los casos de prueba para esto?". Una realimentación concreta sobre el estado actual del sistema realmente no tiene precio. El optimismo es un riesgo del oficio de la programación. La realimentación es el tratamiento.
La realimentación trabaja a diferentes escalas de tiempo. Primero, la realimentación trabaja a escala de minutos y de días. Los programadores escriben pruebas de unidad para toda la lógica en el sistema que podría posiblemente fallar. Los programadores tienen una realimentación precisa minuto-a-minuto sobre el estado de su sistema. Cuando los clientes escriben nuevas "historias", los programadores inmediatamente las estiman, de tal forma que los clientes tienen una realimentación concreta sobre la calidad de sus historias. La persona que controla el progreso, observa la finalización de las tareas, para dar a todo el equipo la realimentación sobre si ellos tienen probabilidad de acabar aquello que tienen que hacer en un lapso de tiempo.

La realimentación también trabaja a escala de semanas y meses. Los clientes y las personas que realizan las pruebas escriben pruebas funcionales de todas las historias implementadas en el sistema. Tienen una realimentación concreta sobre el estado actual de su sistema. Los clientes revisan la planificación cada dos o tres semanas para ver si la velocidad global del equipo es igual a la planificada, y se ajusta el plan. El producto se pone en producción tan pronto como tenga sentido, así el negocio puede comenzar a "sentir" que el sistema está en funcionamiento y descubrir cómo se puede mejorar su exploración.

Las primeras producciones necesitan un poco de explicación. Una de las estrategias en el proceso de planificación es que el equipo ponga las historias valiosas en producción tan pronto como sea posible. Esto da a los programadores una realimentación concreta sobre la calidad de sus decisiones y del proceso de desarrollo.

Muchos proyectos parece que usan la estrategia opuesta. El pensamiento para ir en esta dirección "tan pronto como el sistema esté en producción, no se pueden hacer grandes

cambios 'interesantes', por lo tanto hay que mantener el sistema en desarrollo el mayor tiempo posible".

Esto es exactamente al revés. "En desarrollo" es un estado temporal, aquel en el que el sistema estará solamente un pequeño porcentaje de su vida. Mucho mejor es dar al sistema vida independiente, para que se ponga en pie y respire por sí mismo. Hay que vivir soportando de forma simultánea la producción y el desarrollo de nuevas funcionalidades. Es mejor acostumbrarse a llevar adelante la producción y el desarrollo, cuanto antes mejor.

La realimentación concreta actúa junto con la comunicación y la sencillez. Cuanto mayor realimentación se tenga, más fácil será la comunicación. Si alguien tiene alguna objeción a algún código que se ha escrito y se dispone de un caso de prueba para el que falla, eso es equivalente a miles de horas de discusión sobre la estética. Si se tiene una comunicación clara, se sabrá qué hay que probar y medir para aprender del sistema. Los sistemas sencillos son fáciles de probar. Escribir las pruebas orienta hacia cómo se puede simplificar el sistema, hasta que las pruebas funcionen, no se tiene nada hecho, cuando funcionan todas las pruebas, ya se tiene algo hecho.(Beck, 2000)

7.4 Valentía

La estrategia de diseño de XP se parece al algoritmo del escalador. Se consigue un diseño sencillo, entonces se hace un poco más complejo, después se simplifica y, a continuación, un poco más complejo. El problema con el algoritmo del escalador es alcanzar la posición óptima, donde ningún pequeño cambio pueda mejorar la situación, pero un gran cambio sí podría.

¿Qué excusa se tendrá para desarrollar en situaciones complicadas? VALENTÍA. De vez en cuando, alguien del equipo tendrá una idea disparatada que podría reducir de forma disparatada la complejidad del sistema global. Si tienen valor, lo intentarán. Funcionará (algunas veces). Si tienen valor, lo intentarán poner en producción.

Si no tienen los tres primeros valores, la valentía por sí misma se secará. Sin embargo, combinada con la comunicación, la sencillez y la realimentación concreta, la valentía es extremadamente valiosa.

La comunicación apoya la valentía porque abre la posibilidad para experimentos de alto riesgo y grandes recompensas.

La sencillez apoya a la valentía porque se puede proporcionar mucho más valor con un sistema sencillo, ya que es menos probable estropearlo sin querer. La valentía apoya la sencillez porque tan pronto como se ve la posibilidad de simplificar el sistema, se intenta. La realimentación concreta apoya la valentía porque se siente una mayor seguridad al intentar un cambio radical sobre el código si se puede pulsar un botón y ver que las pruebas funcionan bien (o no, en cuyo caso se vuelve sobre el código otra vez). (Beck, 2000)

8. PRINCIPIOS BÁSICOS

Los cuatro valores (comunicación, sencillez, realimentación y valentía), dan los criterios para una solución satisfactoria. Sin embargo, los valores son demasiado vagos para proporcionar ayuda en decidir qué prácticas seguir. Necesitamos destilar los valores en principios concretos que podamos utilizar.

Estos principios ayudan a escoger entre alternativas. Se prefiere una alternativa que coincida con los principios más plenamente que otra que no lo haga. Cada principio incorpora los valores. Un valor puede ser vago. Lo que es simple para una persona es complejo para otra. Un principio es más concreto. O se tiene realimentación rápida o no se tiene. Los principios fundamentales son:

- Realimentación rápida.
- Asumir sencillez.
- Cambio incremental.
- Aceptar el cambio.
- Trabajar con calidad.

8.1 Realimentación Rápida

La psicología del aprendizaje enseña que el tiempo entre una acción y su realimentación es crítico para aprender.

Los negocios aprenden cómo el sistema puede contribuir más, y realimentar lo que aprenden en días o semanas en vez de en meses o años. Los programadores aprenden a cómo mejorar al diseñar, implementar y probar el sistema, y realimentar lo aprendido en segundos o minutos en vez de días, semanas o meses.

8.2 Asumir la sencillez

Tratar cada problema como si se pudiese resolver de la forma más sencilla.

En muchos casos, éste es el principio más difícil de asumir por los programadores. Se suele hablar tradicionalmente de planificar para el futuro. En cambio, XP propone que se haga un buen trabajo (pruebas, recodificación, comunicación) para resolver el día a día y que se tenga confianza para añadir complejidad en el futuro, cuando se necesite. La economía del software como opción favorece esta aproximación.

8.3 Cambio incremental

Los grandes cambios hechos de una vez no funcionan.

Se encuentran cambios incrementales aplicados de muchas formas en XP. Los cambios de diseño, un poco cada vez. Los cambios de planes, un poco cada vez. Los cambios de equipo, un poco cada vez. La adopción de XP se debe hacer poco a poco.

8.4 Aceptar el cambio

La mejor estrategia es aquella que conserva la mayoría de opciones mientras se resuelve el problema más acuciante.

8.5 Trabajo de calidad

De las cuatro variables de desarrollo de un proyecto (ámbito, coste, tiempo y calidad) la calidad no es realmente una variable independiente. Los únicos posibles valores son "excelente" y "muy excelente". De otra manera no se disfruta del trabajo, no se trabaja bien, y el proyecto va directo a la papelera.

Algunos de los principios menos importantes que todavía nos ayudarán a decidir qué hacer en situaciones específicas son los siguientes:

- Enseñar a aprender.
- Pequeñas inversiones iniciales.
- Jugar a ganar.
- Experimentos concretos.
- Comunicación abierta y honesta.
- Trabajar con los instintos de las personas, no contra ellos.
- Aceptar la responsabilidad.
- Adaptación particular.
- Ir ligero de equipaje.
- Medidas honestas.

9. ROLES DEL PERSONAL

Existen diferentes roles (actores) y responsabilidades en XP para diferentes tareas y propósitos durante el proceso:

9.1 El Programador

El programador es el corazón de XP. En realidad, si los programadores siempre pudiesen tomar decisiones que cuidadosamente equilibrasen las prioridades a corto y largo plazo, no habría necesidad de ningún otro personal técnico en el proyecto distinto a los programadores. Por supuesto, si el cliente realmente no necesitase el software para mantener en funcionamiento el negocio, no habría necesidad de programadores.

En apariencia, ser un programador XP se parece mucho a ser un programador dentro de otros métodos de desarrollo software. Gastan su tiempo trabajando con programas, haciéndolos más grandes, simples y rápidos. Sin embargo, su trabajo no ha finalizado cuando el ordenador entiende que debe hacer. Su principal preocupación es la comunicación con otras personas. Si el programa funciona pero hay algún componente de comunicación esencial que no se ha hecho, el programador no ha acabado.

Los programadores escriben pruebas que demuestren algún aspecto esencial del software. Dividen el programa en partes más pequeñas o combinan partes demasiado pequeñas en una más grande. Intentan desarrollar el software más valioso para el cliente, no desarrollan nada que no sea valioso.

El programador escribe las pruebas unitarias y produce el código del sistema. Es el responsable de las decisiones técnicas y además de sus funciones de programación realiza otras propias de diseño y pruebas.

Hay habilidades que debe poseer un programador XP que no son necesarias o al menos no están tan acentuadas en otros métodos de desarrollo. La programación en parejas es una habilidad que se puede aprender, pero a menudo, encaja bien con las tendencias de la clase de personas que normalmente se dedican a programar. El programador se tendrá que comunicar y coordinar estrechamente con otros programadores a fin de conseguir el éxito.

Otra habilidad necesaria para los programadores XP es el hábito de la simplicidad. Un programador con todos y cada uno de los últimos patrones de análisis y diseño preparados y a mano, probablemente no triunfará con XP.

Un programador tiene que ser capaz de programar razonablemente bien. Tiene que ser capaz de recodificar, que es una herramienta de al menos tanta profundidad y sutileza como

la programación en primera instancia. Tiene que ser capaz de hacer pruebas de unidad al código que, como la recodificación, requiere experiencia y juicio para aplicarlas bien.

Tiene que estar dispuesto a dejar de lado la sensación de propiedad individual de alguna parte del sistema a favor de compartir la propiedad de todo el sistema. Si alguien cambia el código que ha escrito, en cualquier parte del sistema, tiene que confiar en los cambios y aprender de ellos.

9.2 El Cliente

El cliente es otra mitad de la dualidad esencial de XP. El programador sabe cómo programar. El cliente está dispuesto a aprender tanto como el programador.

Ser un cliente XP no es fácil. Hay habilidades que tiene que aprender, como escribir buenas historias, y tener una actitud que le haga triunfar. Sobre todo, tiene que conseguir estar cómodo influyendo en un proyecto con ser capaz de controlarlo. Fuerzas externas a su control darán formas a lo que realmente se consigue construir, tanto como las decisiones que tome. Cambios en las condiciones del negocio, tecnología, composición y la capacidad del equipo, todo esto tiene un gran impacto sobre el software que se entrega.

Un cliente tendrá que tomar decisiones. Ésta es la habilidad más dura para algunos de los clientes. Estaban acostumbrados a que las tecnologías de la información proporcionaran la mitad de lo que prometían y de lo que proporcionaban, la mitad fallaba. Si se es un cliente XP, el equipo necesita que diga con confianza “esto es más importante que eso”, “tanto de esta historia es suficiente”, “estas historias juntas son suficientes”.

Los mejores clientes son aquellos que realmente utilizan el sistema que se está desarrollando, pero que también tienen una cierta perspectiva sobre el problema a resolver.

Resumiendo, un cliente XP escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuales se implementan en cada iteración centrándose en aportar mayor valor al negocio. Es parte del equipo y determina que construir y cuando.

9.3 El Encargado de pruebas (Tester)

Dado que la responsabilidad de muchas pruebas recae sobre las espaldas de los programadores, el rol del encargado de pruebas en un equipo XP está realmente centrado sobre el cliente. Es responsable de ayudar al cliente a escoger y escribir pruebas funcionales. Si las pruebas funcionales no son parte del juego de integración, el encargado

pruebas es el responsable de hacer funcionar las pruebas funcionales regularmente y poner los resultados en un lugar destacado.

Un encargado de pruebas XP no es una persona aislada, dedicada a detectar fallos en el sistema y humillar a los programadores. Sin embargo, alguien tiene que ejecutar las pruebas regularmente (si no se pueden ejecutar juntas las pruebas de unidad y funcionales), difundir los resultados de las pruebas, y asegurarse de que las herramientas de pruebas funcionan bien.

Por tanto, un encargado de pruebas ayuda al cliente a escribir las pruebas funcionales, ejecuta las pruebas regularmente, difunde los resultados en el equipo y es el responsable de las herramientas de soporte para las pruebas.

9.4 El Controlador (Tracker)

Hacer buenas estimaciones es una cuestión de práctica y realimentación. El trabajo de un controlador es cerrar el bucle de la retroalimentación.

También es responsable de echarle un ojo al conjunto del sistema. A mitad de una iteración debería ser capaz de decirle al equipo lo que ellos harán, si seguirán el curso actual o si necesitan cambiar algo. En un par de iteraciones en una planificación, debería ser capaz de decirle al equipo lo que ellos van a crear en la siguiente versión sin hacer grandes cambios.

El controlador es el historiador del equipo. Guarda un diario de las puntuaciones de las pruebas funcionales. Guarda un diario de los defectos comunicados, quién acepta la responsabilidad para cada uno, y qué casos de prueba se añadieron a cada uno de dichos defectos.

Resumiendo, el controlador proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del proceso de cada iteración. Observa son molestar y conserva datos históricos.

9.5 El Preparador (Coach)

Como preparador, es el responsable del proceso en su conjunto. Observa cuándo la gente se está desviando del proceso del equipo. Permanece en calma cuando todo el mundo tiene pánico.

Todo el mundo en un equipo XP es responsable de entender la aplicación de XP en algún punto. El preparador es responsable de entenderla con mucha más profundidad: que

prácticas alternativas pueden ayudar al conjunto actual de problemas; cómo están utilizando otros equipos XP; qué ideas subyacen en XP; y cómo relacionan con la situación actual.

Si el preparador ve un error en el diseño, primero tiene que decidir si es lo suficientemente importante como para que intervenga. Cada vez que guía al equipo, hace que ellos sean mucho menos independientes. Demasiada dirección da lugar a una baja productividad, una baja calidad y una baja moral. Por tanto, lo primero que tiene que decidir es si el problema que ve es de tal envergadura como para que necesite aceptar el riesgo de intervenir. Si decide que realmente sabe más que el equipo, entonces tiene que hacer una observación tan discreta como sea posible. Otras veces, debe ser directo.

El rol del preparador disminuye conforme madura el equipo. De acuerdo con los principios de control distribuido y responsabilidad aceptada "el proceso" sería responsabilidad de todos.

9.6 El Consultor

Lo más probable es que un consultor no quiera ser utilizado para un trabajo extremo. Es probable que vea que el equipo nuestra una cierta cantidad de escepticismo. Pero el equipo debería ser extremadamente claro sobre el problema que necesitan resolver.

Un consultor es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.

9.7 El Gran Jefe (Big Boss)

Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

Lo que más necesita un equipo de un gran jefe es coraje, confianza e insistencia de forma ocasional en que ellos hagan lo que dicen que hacen.

10. CUÁNDO NO SE DEBERÍA INTENTAR XP

Es importante no utilizar XP donde esté destinada a fracasar, como es importante utilizarla donde proporcione ventajas claras.

La mayor barrera para el éxito de un proyecto XP es la cultura. No la cultura nacional, aunque eso también tiene un efecto, sino la cultura empresarial.

Si un cliente o un director de proyecto insiste en una especificación o análisis o diseño completo antes de que comience la pequeñez de la programación, entonces aparecerá seguro una fricción entre la cultura del equipo y la del cliente o director de proyecto. Todavía el proyecto puede funcionar bien utilizando XP, pero no será fácil.

Otra cultura que no es adecuada para XP es aquella en la que se pide trabajar durante bastantes horas para demostrar el compromiso con la empresa. No se puede llevar a cabo XP cansado. Si la cantidad producida por un equipo que trabaja a velocidad tope no es suficiente para la empresa, entonces XP no es la solución. Una segunda semana consecutiva de horas extraordinarias para un proyecto XP, es una señal inequívoca de que algo no funciona en el proceso, y se debería corregir ese error.

Hasta los programadores inteligentes tienen momentos difíciles con XP. Algunas veces el personal inteligente tiene el momento más difícil negociando el juego de "suposición correcta" para la comunicación estrecha y la evolución continua.

El tamaño evidentemente es importante. Probablemente no se podrá poner en marcha un proyecto XP con cien programadores. Ni con cincuenta. Ni probablemente con veinte. Diez es factible, sin duda. La cantidad de funcionalidad a producir y el número de personas que la producen, no tiene ninguna clase de relación lineal simple.

Otra barrera para XP es un entorno donde se necesita mucho tiempo para obtener la realimentación. Si se tiene un entorno físico inapropiado, XP no funciona. Una gran habitación con pequeños cubículos alrededor y máquinas potentes en mesas situadas en medio de la habitación, es el mejor entorno.

11. DESCRIPCIÓN GLOBAL DEL PROYECTO

Este proyecto tiene como objetivo la evaluación y aplicación de las metodologías agiles, en concreto de la metodología Extreme Programming o Programación Extrema a un caso práctico.

El estudio se basa en demostrar mediante el desarrollo de un proyecto de gestión de un restaurante que la aplicación de esta metodología mediante unos patrones de productos es más eficiente que sin utilizar dichos patrones.

Con el fin de que el hecho de haber tenido que aprender XP no interfiera en el resultado del estudio, hemos incluido una fase inicial, o fase cero, en la que hablaremos del aprendizaje de dicha metodología.

En una primera fase del proyecto se han realizado las historias de usuarios, la estimación y los casos de prueba para la parte de las reservas de mesas del restaurante en cuestión, tomando tiempos de realización para el posterior estudio.

Esta primera fase se ha logrado con la búsqueda de información en torno a XP que existe en la bibliografía, con esto hemos querido semejar el trabajo de un desarrollador que aplica esta metodología por primera vez.

Una segunda fase del proyecto ha sido la realización de los patrones de producto que posteriormente se aplicarán en la siguiente fase para la parte de los pedidos de consumiciones del restaurante.

Un patrón describe un problema que ocurre una y otra vez en nuestro entorno, describiendo el núcleo de la solución a dicho problema, de tal forma que se puede usar esta solución varias veces sin necesidad de hacerlo dos veces de la misma forma.

Por otra parte el uso de patrones para el desarrollo de software establece la diferencia entre un buen y un mal diseño. Un patrón es un fragmento nombrado de información instructiva, que captura la estructura esencial y la visión interna de una familia de soluciones con probado éxito sobre un problema recurrente que surge dentro de un cierto contexto y fuerzas de sistema. En otras palabras, rehusar soluciones que funcionaron bien una vez.

Una vez se hayan realizado los patrones de producto que llevaremos a cabo, la tercera parte de este proyecto se basa en la aplicación de estos patrones al caso práctico nombrado anteriormente, pero para la parte de los pedidos de las consumiciones de los comensales. Con esta aplicación lo que se ha querido plasmar ha sido que la aplicación de dichos patrones hace que el diseño del software se hace más eficiente y con mayor rapidez, para ello también se han tomado los tiempos que ha llevado hacer esta parte.

Una vez terminada la fase tercera hemos analizado los tiempos de esta misma fase y de la primera realizada con anterioridad. Este estudio nos ha demostrado lo que ya se ha dicho anteriormente, es decir, que la aplicación de los patrones hace que la aplicación de la metodología ágil XP sea más rápida y eficiente.

12. CASO PRÁCTICO PARA LA APLICACIÓN DE XP

En este apartado describiremos la especificación de requisitos del caso práctico que utilizaremos para la aplicación de la metodología de extreme programming. Este caso se basa en una cadena de restaurantes, y su especificación de requisitos es la que mostramos a continuación:

12.1 Descripción del Caso

Una cadena de restaurantes quiere automatizar el proceso de reservas así como el de los pedidos de cada mesa y la cantidad que hay en la cocina de cada unos de los productos que se manejan para la realización de cada plato, y que obviamente han de ser repuestos desde el almacén a medida que estos se van terminando.

Reservas de Mesas

Los clientes de los restaurantes pueden llamar por teléfono para reservar una mesa, pero lo que se está intentando poner de moda es el uso de unos terminales punto de reserva (TPR) ubicados en la calle. La ventaja que tiene el uso de estos terminales es la posibilidad de elegir la mesa en función de su ubicación dentro del restaurante, cosa que no se puede hacer por teléfono.

Todos los TPR son de la cadena de restaurantes, aunque cabe la posibilidad de que en un futuro, distintas cadenas de restaurantes puedan ofrecer sus servicios a través de estos terminales. Hoy por hoy sólo se podrán elegir restaurantes de esta cadena de restaurantes.

Cuando un cliente se conecta a uno de estos TPR, el terminal le pregunta en que restaurante quiere realizar la reserva, qué día y la hora. EL terminal, con ayuda del Centro de Reservas, que tiene información del estado de todas las mesas de los restaurantes de la cadena, comprueba si en el restaurante especificado hay alguna mesa libre a esa hora. Si es así, el centro de reservas con ayuda del restaurante en cuestión, envía al TPR en primer lugar un plano del restaurante y a continuación las mesas que están libres ubicadas en el lugar del plano que las corresponde. Así el TPR puede reconstruir el plano del restaurante con las mesas que hay libres.

Las mesas están separadas en mesas de fumador, marcada con la F y de no fumador marcadas con NF. Además cada mesa lleva un indicador con el número de personas para el que está pensada dicha mesa.

El usuario selecciona una mesa e indica el número de personas que van a ocuparla, el TPR se lo comunica a la central de reservas que a su vez confirma con el restaurante que efectivamente todo sigue en orden. Si todo está bien, el terminal pide al usuario que indique

el nombre con el cual desea realizar la reserva, el usuario se lo indica y el terminal se lo comunica al Centro de Reservas el cual hace la reserva por él y le da un ticket indicando el día, la hora, la mesa y el nombre con el que ha reservado la mesa.

Si el cliente llega al restaurante 20 minutos después de la hora de reserva de la mesa, el sistema se encargará automáticamente de dejar libre dicha mesa.

Si no hay mesas libres a la hora indicada por el usuario, el TPR se lo comunica al cliente, dándole además la posibilidad de solicitar al sistema sugerencias sobre restaurantes disponibles a la hora y en el día solicitado. En el caso en que el cliente quiera una sugerencia del sistema, el Centro de Reservas le facilita al Cliente a través del TPR una lista de posibles restaurantes. El usuario podrá seleccionar alguno, en cuyo caso el procedimiento es el mismo que para el caso de la reserva normal exceptuando que el TPR ya tiene ciertos datos del cliente.

Si lo que ocurre es que sí hay mesas, pero el cliente no encuentra ninguna mesa que le satisfaga a la hora a la que desea la reserva, puede solicitar al sistema que le indique otro restaurante de la cadena que también tenga mesas libres a esa hora.

Si en cualquiera de los casos el usuario cambia de idea, basta con que cancele en cualquier momento la operación.

Cuando un cliente llega a uno de los restaurantes de la cadena, se le pregunta si tiene reserva o no.

En el caso en que tenga reserva, bastará con que presente el ticket, si la hora de reserva no supera en 20 minutos a la hora de llegada al restaurante, la mesa pasa de estar reservada a ocupada y se les sienta en el lugar que les corresponde.

Si por el contrario la hora de llegada supera en 20 minutos a la hora de reserva, el sistema se habrá encargado de anular dicha reserva de modo que la mesa haya quedado libre para otro posible cliente, por tanto se les trata del mismo modo que si no tuvieran reserva. En ese caso el encargado en ese momento de las reservas, solicita al sistema que le muestre las mesas libres para ese momento, si hay mesas libres, le pregunta al usuario si quiere mesa de fumador o de no fumador y cuantas personas son, el usuario se lo dice y en caso de que haya mesa libre, el encargado hace la reserva y les sienta. Si no hay mesa el encargado le debe pedir al sistema el tiempo aproximado para que quede libre la próxima mesa de las características de la mesa solicitada. Esto podrá calcularlo el sistema a través del estado en que se encuentran las distintas mesas en un determinado momento, estos estados son:

- Libre: si nadie la ha reservado
- Reservada: si alguien ha hecho una reserva
- Ocupada: si los comensales están ya a la mesa

- Pidiendo: si el camarero está recogiendo el pedido de esa mesa
- En espera de comida: si están esperando que se les sirva
- Servidos: si los comensales ya tienen la comida en la mesa
- Esperando cuenta: si los comensales hay pedido la cuenta
- Pagando: si los comensales ya tienen la cuenta en la mesa

Además si no hay mesas libres y el cliente lo desea, se le debe informar de otro/s restaurante de la cadena que sí tenga mesas libres.

Pedidos

Una vez que los clientes están a la mesa, los camareros les dan la carta y esperan que pidan. Los camareros tienen unos dispositivos que controlan una parte del sistema, el de los pedidos en cada mesa.

Esta parte del sistema está a la espera de que el camarero introduzca un número de mesa.

Cuando el camarero introduce el número de la mesa que va a pedir, se graba automáticamente la hora del pedido y la mesa que lo está haciendo. Los clientes pueden pedir tanto comidas como bebidas, ambas se consideran consumiciones. Cada tipo de consumición tiene un código que será lo que el camarero introduzca en el sistema.

Si un cliente quiere saber los ingredientes de un determinado plato se lo puede preguntar al camarero el cual a su vez lo consulta al sistema, tecleando el código de la consumición seguido del símbolo de interrogación.

El pedido de cada mesa se va componiendo de líneas de pedido donde cada línea de pedido es una consumición. Es decir, si se piden tres platos de pasta y dos cervezas, el pedido tendrá cinco líneas de pedido.

El camarero introduce por cada consumición el código de esta y pulsa aceptar, antes de poder volver a introducir un código de consumición, el sistema debe ser capaz de comprobar que hay ingredientes necesarios para satisfacer dicha petición de consumición.
Si no fuera el caso, es decir si no se pudiera completar la consumición por falta de uno o varios ingredientes, el camarero indicará al cliente que no es posible para que pida otra cosa. Por supuesto al detectarse esta situación se debe informar al almacén de que reponga cada uno de los ingredientes o bebidas que faltan.

Una vez que los comensales terminan de pedir, el camarero cierra temporalmente la nota e decir pulsa fin, mientras no le pidan nada más y la mesa pasa a estar en estado de "Esperar comida". Automáticamente el sistema avisa en cocina que hay un nuevo pedido en una mesa determinada. En este momento, se recorre cada línea del pedido, de nuevo, para ir a su vez recorriendo los ingredientes de cada consumición y disminuir la cantidad que se

tiene de un determinado producto en cocina, de modo que si la cantidad del producto disminuye por debajo del umbral establecido para ese alimento, se pida automáticamente a almacén.

El encargado de la cocina observa cuando llega un nuevo pedido y se lo indica a los cocineros. Cuando los platos están listos el encargado de cocina, establece el pedido de esa mesa como cocinado y manda un mensaje al control del camarero para que recoja el pedido de la mesa indicada, el camarero lo recoge para llevarlo a la mesa que corresponde e indica que esa mesa está servida.

Control de ingredientes

Además, como ya señalábamos antes, desde la cocina también se lleva el control de los ingredientes, como se sabe exactamente los ingredientes de cada plato, una vez se ha preparado la/s bandejas que contienen el pedido de una mesa, se indica al sistema que los ingredientes que contenían esos platos o consumiciones han disminuido de modo que cuando rebasan el mínimo indispensable en cocina, el sistema avisa automáticamente para que repongan desde almacén.

Pago y liberación de mesas

Cuando los comensales han terminado, piden al camarero la nota, momento en el cual el camarero cierra definitivamente el pedido de esa mesa y establece el estado de la mesa como esperando nota. El camarero ordena que se imprima la nota que está compuesta por cada una de las líneas de pedido. Una vez está impresa se la pasa a los clientes y éstos depositan bien el dinero en efectivo o una tarjeta. El camarero se va a la caja central e indica que esa mesa está pagando, vuelve con la nota cobrada, y establece la mesa como libre.

12.2 Fase Cero: Aprendizaje de la Metodología XP

Una de las partes a la que le hemos dedicado más tiempo ha sido en el aprendizaje de la metodología, ya que, necesitábamos este conocimiento para la realización de las fases que más tarde expondremos.

Principalmente, la razón por la que debíamos aprender XP es porque, la primera fase de este estudio se basa en que un desarrollador sin experiencia empiece a utilizar esta metodología.

Lo primero que cualquier persona hará para saber que debe hacer en cualquier situación, será documentarse sobre el tema a manejar. Esto es lo que hemos tratado de asemejar.
Los dos libros más consultados son los descritos en la bibliografía del proyecto

Esta bibliografía ha sido de gran ayuda para la comprensión de las metodologías Ágiles, pero sobre todo la metodología XP, en que se basa, que utiliza, como se lleva a cabo…
Además de los libros mencionados antes, se han encontrado una serie de documentos, artículos, etc vía internet que sobre todo ha ayudado a dar una visión más real de la utilización de la metodología ya que en algunos casos son puntos de vista de diseñadores, desarrolladores,…

Aunque nos hemos dado cuenta de que prácticamente todos los documentos se basan en puntos de vista de Kent Beck, con lo que llegan a decir lo mismo.
El aprendizaje y manejo de esta metodología nos ha resuelto gran cantidad de incertidumbres en la realización de las actividades necesarias para el estudio de este proyecto, como por ejemplo cómo y para qué se realizan historias de usuario, quien las realiza, como se estiman, etc.

12.3 Fase primera: Aplicación de XP al Caso Práctico

Lo primero que se ha realizado para la aplicación de la metodología XP ha sido la búsqueda de información acerca de cómo funciona y cómo se utiliza dicha metodología. Para ello se ha utilizado como herramienta la búsqueda online de información y la visualización de la bibliografía que más adelante detallaremos.

Una vez tenemos claro como deberemos llevar a cabo la metodología, se ha dividido el caso del restaurante en dos partes, la de reservas y la de pedidos. Con la parte que trabajaremos en ésta fase será con la de reservas.

Lo que realizamos con esta parte será el estudio de la aplicación de la metodología con la información que hemos recogido en la búsqueda nombrada anteriormente.

Esta es una de las partes que ha llevado más tiempo, ya que aunque en la primera parte del proyecto hemos hablado de la reutilización, en general no se ha encontrado gran cosa como ayuda para un programador sin experiencia en este ámbito, por lo que una vez encontrados unos documentos donde detallaban a groso modo como comenzar con nuestro estudio, empezamos a aplicar dichas doctrinas, que sería lo que en realidad haría un programador Junior.

En la puesta en marcha de nuestro estudio se han realizado las historias de usuario, la estimación y pruebas para esa parte del caso donde se habla de las reservas de mesas.

Para la realización de esta parte se han utilizado las tablas que anteriormente se han expuesto, con el fin de hacer este trabajo un poco más sencillo ya que no se han encontrado documentos de ayuda en la bibliografía que más adelante se detalla.

Una de las particularidades que hemos desarrollado ha sido en la estimación de las historias de usuario, ya que al realizar estas estimaciones con puntos de historias y al no tener doctrinas que nos indiquen porque una historia debe tener más puntos de historia que otra, hemos realizado una priorización de dichas historias basándonos en dos variables que más tarde detallaremos con más exactitud.

12.3.1 Realización de historias de usuarios

Para la elaboración de las historias de usuarios hemos identificado las actividades que se realizan para la parte de las reservas de mesa en el restaurante, y para cada una de ellas hemos ido rellenando la tabla de historias de usuario.

Lo primero que hemos identificado es el “número”, el “nombre” de la historia y el “usuario” que realiza la acción.

Una vez identificadas todas estas variables, el segundo paso sería rellenar el campo "Descripción". Este campo da una explicación de qué es lo que se realiza en esa acción a groso modo, ya que, no debemos olvidar que las historias de usuario son realizadas por los clientes, y que en muchos casos estos no saben cómo ni qué quieren realmente. Se refiere a que es lo que el cliente quiere que se realice en esa acción, y como quiere que se lleve a cabo.

Una vez claro que es lo que se quiere y como se quiere iremos al campo observaciones en el que se detallarán los puntos que no habremos detallado anteriormente, solo si los hubiera.

Una vez tengamos todos esos datos se dará valor a la priorización y al riesgo para cada una de las historias de usuario.

La priorización ha sido en base a la importancia de la historia en nuestro negocio de restaurante, es decir, podríamos pensar de esta manera, si no existiera es historia, qué podríamos perder, cuánto de innovador es la historia en la empresa…
En cambio, el riesgo se ha medido en base a la complejidad de realización de esta historia. Por lo que una historia que sea más compleja de realizar y que conlleve muchas subtareas internas tendrá un riesgo más alto que una historia más simple y que solo conlleve una o dos tareas internas.

A continuación, se exponen las historias de usuario realizadas en esta primera fase del proyecto, y siguiendo los pasos anteriormente descritos:

1. HACER RESERVA_TELÉFONO

<table>
<tr><th colspan="3">Historia de Usuario</th></tr>
<tr><td>Número: 1</td><td colspan="2">Nombre: Hacer reserva_teléfono</td></tr>
<tr><td colspan="3">Usuario: CLIENTE</td></tr>
<tr><td colspan="2">Modificación de Historia Número:</td><td>Iteración Asignada:</td></tr>
<tr><td colspan="2">Prioridad en Negocio: Bajo

(Alta / Media / Baja)</td><td>Puntos Estimados:</td></tr>
<tr><td colspan="2">Riesgo en Desarrollo: Bajo

(Alto / Medio / Bajo)</td><td>Puntos Reales:</td></tr>
<tr><td colspan="3"></td></tr>
</table>

Descripción: Los clientes del restaurante pueden llamar por teléfono para realizar la reserva de una mesa. Se seguirá el mismo procedimiento que el caso de realizar la reserva por tpr, solo que no se podrá elegir mesa en función de la ubicación dentro del restaurante.
Observaciones:

2. HACER RESERVA_TPR:

Historia de Usuario	
Número: **2**	Nombre: **Hacer reserva_TPR**
Usuario: CLIENTE	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: Alto (Alto / Medio / Bajo)	Puntos Reales:
Descripción: El cliente se conecta e introduce en el sistema el nombre del restaurante, el día y la hora a la que desea hacer la reserva. El restaurante con los datos recibidos comprueba la disponibilidad de mesas construyendo un plano del restaurante, ubicando las mesas libres. El cliente selecciona la mesa que más se adecue a sus preferencias e indica el número de comensales que la van a ocupar. Además el cliente deberá introducir el nombre con el que desea hacer la reserva. Finalmente el sistema imprimirá un ticket que confirma que la reserva ha sido realizada con éxito.	
Observaciones:	

3.CASO ALTERNATIVO 1. HACER RESERVA_TPR

Historia de Usuario	
Número: **3**	Nombre: **Caso alternativo1:Hacer reserva_TPR**
Usuario: CLIENTE	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Si el cliente selecciona un restaurante en el que no hay mesas libres para la fecha y hora indicada el terminal se lo comunica dándole la posibilidad de solicitar sugerencias sobre restaurantes disponibles para el día y hora requerida. En el caso de que el usuario solicite dicha ayuda se le proporcionará una lista de restaurantes para que proceda a seleccionar uno, siguiendo con el procedimiento de reserva de mesa a través del terminal TPR.	
Observaciones:	

4. CASO ALTERNATIVO 2. HACER RESERVA_TPR

Historia de Usuario	
Número: **4**	Nombre: **Caso alternativo2: Hacer reserva_TPR**
Usuario: CLIENTE	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto	Puntos Estimados:

(Alta / Media / Baja)	
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Si el cliente a la hora de realizar la reserva no encuentra ninguna mesa que satisfaga sus necesidades podrá solicitar al sistema que le indique otro restaurante que tenga mesas libres para el día y hora seleccionada, pudiendo realizar la reserva de la mesa que elija de forma normal.	
Observaciones:	

5.ACTIVAR RESERVA_TPR

Historia de Usuario	
Número: **5**	Nombre: **activar reserva_TPR**
Usuario: CAMARERO	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: Bajo (Alto / Medio / Bajo)	Puntos Reales:
Descripción: El cliente al llegar al restaurante en el que se ha realizado la reserva deberá presentar el ticket entregado cuando ésta se realizó. Si la hora a la que ha llegado no sobrepasa los 20 minutos de la hora reservada el cliente ocupara la mesa correspondiente.	
Observaciones:	

6.CASO ALTERNATIVO 1: ACTIVAR RESERVA_TPR

Historia de Usuario	
Número: **6**	Nombre: **Caso alternativo1:Activar reserva_TPR**
Usuario: CAMARERO	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: Bajo (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Si la hora de llegada del cliente al restaurante sobrepasa los 20 minutos de la hora en la que se ha reservado el sistema habrá anulado la reserva, dejando la mesa libre para otro posible cliente.	
Observaciones:	

7.HACER RESERVA_MAITRE

Historia de Usuario	
Número: **7**	Nombre: **Hacer reserva_Maitre**
Usuario: MAITRE	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados:

Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Cuando un cliente sin reserva o un cliente que tenia reserva, pero ha sido anulada por sobrepasar los 20 minutos de dicha reserva quiere una mesa, el maitre solicita al sistema un plano con las mesas libres, si existen, introduce los datos proporcionados por el cliente para una nueva reserva (numero de comensales, fumador/no fumador…).	
Observaciones:	

8.CASO ALTERNATIVO 1. HACER RESERVA_MAITRE

Historia de Usuario	
Número: **8**	Nombre: **Caso alternativo 1:Hacer reserva_Maitre**
Usuario: MAITRE	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Medio (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: Alto (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Si la reserva ha sido anulada el maître comprueba a través del terminal si aun existen mesas libres. Si no es así pedirá al sistema el tiempo aproximado para que una quede libre. Si el cliente lo desea se deberá informar de otros restaurantes de la cadena que sí cuenten con mesas libres.	
Observaciones:	

Una vez que hemos cumplimentado las tablas de las historias de usuario, el siguiente paso será rellenar la celda de "Puntos Estimados". Para ello deberemos estimar cada historia de usuario. En este caso hemos decidido utilizar puntos de historia, ya que consideramos que es el mejor método para la estimación de una historia de usuario, porque no se basa en tiempos concretos, que para un desarrollador sénior podría no ser el mismo que para uno junior, por lo que usando la estimación por puntos de historia este problema desaparecería porque una historia estimada en tres puntos de historia lo será tanto para un desarrollador sénior como uno junior.

12.3.2 Estimación de las historias de usuario

Como no se tenía ningún método concreto que nos diera pistas sobre porque una historia de usuario podría tener más puntos de historia que otra, hemos hecho una aproximación del método utilizando la priorización de estas historias.

La priorización de historias se ha hecho en base a dos variables:

1. Complejidad de la historia de usuario, a la cual le hemos dado un valor de 0.25

2. Importancia en el negocio, que le hemos concedido un valor de 0.75 considerando que este valor es mucho más importante al tratarse de un negocio de hostelería en el que necesitamos obtener beneficios.

Una vez que tenemos identificadas las variables por las cuales vamos ha realizar la priorización y por consiguiente la estimación, le daremos un valor a cada variable por cada una de las historias de usuario.

Más tarde, cuando tengamos todos los valores correspondientes a todas las historias y todas las variables, haremos unos cálculos que nos darán la priorización de cada historia. Estos cálculos se basan en multiplicar cada valor de las variables dadas a cada historia por el valor global que le hemos dado a cada variable de priorización (0.25 o 0.75 dependiendo de la variable), y después se suman dichos valores.
Cuando tengamos la priorización de las historias, para obtener el valor de los puntos de historia, que es lo que realmente queremos obtener, realizaremos un cómputo basado en que la historia que mayor priorización tenga es la que tendrá el valor más alto de puntos de historia. En nuestro caso, hemos tomado dicho valor en 5 puntos de historia, y en base a esto se irán hallando los demás valores.

A continuación se expondrán la priorización de las historias y por consiguiente la estimación de las mismas tal y como se ha explicado anteriormente pero de una forma más visual y cómoda.

Variables de historia:

A: Complejidad de historia (0.25)

B: Importancia en el negocio (0.75)

HISTORIA	A	B	TOTAL	PUNTOS DE HISTORIA
1. Hacer reserva teléfono	4	3	4*0.25+3*0.75=3.25	2
2.Hacer reserva TPR	6	10	6*0.25+10*0.75=9	5
3. CA1: Hacer reserva TPR	6	9	6*0.25+9*0.75=8.25	4
4. CA2: Hacer reserva TPR	6	9	6*0.25+9*0.75=8.25	4
5. Activar reserva TPR	4	10	4*0.25+10*0.75=8.5	5
6. CA1: Activar reserva TPR	4	8	4*0.25+8*0.75=7	4
7. Hacer reserva maître	6	10	6*0.25+10*0.75=9	5
8. CA1: Hacer reserva maître	9	7	9*0.25+7*0.75=7.5	4

Una vez tenemos la estimación terminada, rellenamos la celda de las tablas de historias de usuario con estos datos como se exponen a continuación:

1. HACER RESERVA_TELÉFONO

Historia de Usuario	
Número: **1**	Nombre: **Hacer reserva_teléfono**
Usuario: CLIENTE	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Bajo (Alta / Media / Baja)	Puntos Estimados: 2
Riesgo en Desarrollo: Bajo (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Los clientes del restaurante pueden llamar por teléfono para realizar la reserva de una mesa. Se seguirá el mismo procedimiento que el caso de realizar la reserva por tpr, solo que no se podrá elegir mesa en función de la ubicación dentro del restaurante.	
Observaciones:	

2. HACER RESERVA_TPR:

Historia de Usuario	
Número: **2**	Nombre: **Hacer reserva_TPR**
Usuario: CLIENTE	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados: 5

Riesgo en Desarrollo: Alto (Alto / Medio / Bajo)	Puntos Reales:
Descripción: El cliente se conecta e introduce en el sistema el nombre del restaurante, el día y la hora a la que desea hacer la reserva. El restaurante con los datos recibidos comprueba la disponibilidad de mesas construyendo un plano del restaurante, ubicando las mesas libres. El cliente selecciona la mesa que más se adecue a sus preferencias e indica el número de comensales que la van a ocupar. Además el cliente deberá introducir el nombre con el que desea hacer la reserva. Finalmente el sistema imprimirá un ticket que confirma que la reserva ha sido realizada con éxito.	
Observaciones:	

3. CASO ALTERNATIVO 1. HACER RESERVA_TPR

Historia de Usuario	
Número: **3**	Nombre: **Caso alternativo1:Hacer reserva_TPR**
Usuario: CLIENTE	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados: 4
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Si el cliente selecciona un restaurante en el que no hay mesas libres para la fecha y hora indicada el terminal se lo comunica dándole la posibilidad de solicitar sugerencias sobre restaurantes disponibles para el día y hora requerida. En el caso de que el usuario solicite dicha ayuda se le proporcionará una lista de restaurantes para que proceda a seleccionar uno, siguiendo con el procedimiento de reserva de mesa	

a través del terminal TPR.
Observaciones:

4. CASO ALTERNATIVO 2. HACER RESERVA_TPR

Historia de Usuario	
Número: **4**	Nombre: **Caso alternativo2: Hacer reserva_TPR**
Usuario: CLIENTE	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados: 4
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Si el cliente a la hora de realizar la reserva no encuentra ninguna mesa que satisfaga sus necesidades podrá solicitar al sistema que le indique otro restaurante que tenga mesas libres para el día y hora seleccionada, pudiendo realizar la reserva de la mesa que elija de forma normal.	
Observaciones:	

5. ACTIVAR RESERVA_TPR

Historia de Usuario	
Número: **5**	Nombre: **activar reserva_TPR**
Usuario: CAMARERO	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados: 5
Riesgo en Desarrollo: Bajo (Alto / Medio / Bajo)	Puntos Reales:
Descripción: El cliente al llegar al restaurante en el que se ha realizado la reserva deberá presentar el ticket entregado cuando ésta se realizó. Si la hora a la que ha llegado no sobrepasa los 20 minutos de la hora reservada el cliente ocupara la mesa correspondiente.	
Observaciones:	

6. CASO ALTERNATIVO 1: ACTIVAR RESERVA_TPR

Historia de Usuario	
Número: **6**	Nombre: **Caso alternativo1:Activar reserva_TPR**
Usuario: CAMARERO	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto	Puntos Estimados: 4

(Alta / Media / Baja)	
Riesgo en Desarrollo: Bajo (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Si la hora de llegada del cliente al restaurante sobrepasa los 20 minutos de la hora en la que se ha reservado el sistema habrá anulado la reserva, dejando la mesa libre para otro posible cliente.	
Observaciones:	

7. HACER RESERVA_MAITRE

Historia de Usuario	
Número: **7**	Nombre: **Hacer reserva_Maitre**
Usuario: MAITRE	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados: 5
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Cuando un cliente sin reserva o un cliente que tenia reserva, pero ha sido anulada por sobrepasar los 20 minutos de dicha reserva quiere una mesa, el maître solicita al sistema un plano con las mesas libres, si existen, introduce los datos proporcionados por el cliente para una nueva reserva (número de comensales, fumador/no fumador…).	
Observaciones:	

8. CASO ALTERNATIVO 1. HACER RESERVA_MAITRE

<table>
<tr><th colspan="3">Historia de Usuario</th></tr>
<tr><td>Número: 8</td><td colspan="2">Nombre: Caso alternativo 1:Hacer reserva_Maitre</td></tr>
<tr><td colspan="3">Usuario: MAITRE</td></tr>
<tr><td colspan="2">Modificación de Historia Número:</td><td>Iteración Asignada:</td></tr>
<tr><td colspan="2">Prioridad en Negocio: Medio
(Alta / Media / Baja)</td><td>Puntos Estimados: 4</td></tr>
<tr><td colspan="2">Riesgo en Desarrollo: Alto
(Alto / Medio / Bajo)</td><td>Puntos Reales:</td></tr>
<tr><td colspan="3">Descripción: Si la reserva ha sido anulada el maître comprueba a través del terminal si aún existen mesas libres. Si no es así pedirá al sistema el tiempo aproximado para que una quede libre.

Si el cliente lo desea se deberá informar de otros restaurantes de la cadena que sí cuenten con mesas libres.</td></tr>
<tr><td colspan="3">Observaciones:</td></tr>
</table>

Pasamos ahora a otra de las actividades que realizaremos en esta primera fase, se trata de los casos de prueba para cada historia de usuario. Para ello, utilizaremos la tabla presentada en el apartado de pruebas descrito con anterioridad.

Esta actividad se utiliza para comprobar que todo se ha realizado correctamente y que se desempeñan todas las tareas que el cliente desea para cada actividad dentro del sistema. En ella se describen más en detalle las tareas que deben llevarse a cabo en cada una de las historias de usuario, las condiciones que deben haber ocurrido anteriormente para que se de dicha historia de usuario y el resultado esperado para cada una de ellas.

Gracias a estas pruebas, podemos comprobar que el sistema funciona correctamente según las especificaciones de requisitos que nos han llegado gracias a la colaboración continua del

cliente en la metodología. Estos casos de prueba son una de las partes más importantes de la metodología ya que gracias a ellos podemos darnos cuenta de fallos del sistema antes de llegar a la implementación completa, ya que si no fuera así , en fases más tardías tendríamos que volver al principio para realizar cambios, cosa que no es fácil.

La metodología XP se basa en comprobaciones continuas del funcionamiento del sistema. Si se encontrasen errores se podrían hacer cambios en fases tempranas del diseño del sistema. Esto es mucho más fácil cuanto antes se haga, y como estas comprobaciones se hacen continuamente no tenemos el problema de encontrarnos demasiados errores más tarde ya que estos se van solucionando poco a poco en las comprobaciones diarias.

A continuación expondremos los casos de prueba aplicados a las historias de usuario del caso de prueba del restaurante. Al igual que para las historias de usuarios nos referiremos a los casos de prueba para la parte de reservas.

1. Caso de prueba 1: Hacer reserva teléfono

<table>
<tr><th colspan="2">Caso de Prueba de Aceptación</th></tr>
<tr><td>Código: 1</td><td>Historia de Usuario (Nro. y Nombre): Hacer reserva_teléfono</td></tr>
<tr><td colspan="2">Nombre: Hacer reserva_teléfono</td></tr>
<tr><td colspan="2">Descripción: Los clientes del restaurante pueden llamar por teléfono para realizar la reserva de una mesa. Se seguirá el mismo procedimiento que el caso de realizar la reserva por tpr, solo que no se podrá elegir mesa en función de la ubicación dentro del restaurante.</td></tr>
<tr><td colspan="2">Condiciones de Ejecución:
1. El cliente llama por teléfono.</td></tr>
<tr><td colspan="2">Entrada / Pasos de ejecución:
1. El cliente llama por teléfono para realizar una reserva de una mesa en el restaurante.
2. El camarero le pregunta al cliente en qué restaurante quiere efectuar la reserva, que día y a qué hora.
3. El camarero comprueba si en el restaurante especificado por el cliente hay mesas libres para esa hora y día.
4. Si es así, el camarero le pregunta al cliente que tipo de mesa desea y cuantos</td></tr>
</table>

comensales serán.
5. El camarero selecciona una mesa con las preferencias del cliente e indica el número de comensales que van a ocuparla.
6. El camarero se lo comunica a la central de reservas que a su vez confirma con el restaurante que todo está correcto.
7. Si todo está bien, el camarero pide al usuario el nombre con el que desea realizar la reserva.
8. El camarero se lo comunica al Centro de Reservas, el cual hace la reserva cambiando el estado de la mesa a "reservada".

Resultado Esperado: Reserva de una mesa en el restaurante

Evaluación de la Prueba:

2. Caso de prueba 2: Hacer reserva_tpr

Caso de Prueba de Aceptación	
Código: 2	**Historia de Usuario (Nro. y Nombre): Hacer reserva_tpr**
Nombre: Hacer reserva_tpr	
Descripción: El cliente se conecta e introduce en el sistema el nombre del restaurante, el día y la hora a la que desea hacer la reserva. El restaurante con los datos recibidos comprueba la disponibilidad de mesas construyendo un plano del restaurante, ubicando las mesas libres. El cliente selecciona la mesa que más se adecue a sus preferencias e indica el número de comensales que la van a ocupar. Además el cliente deberá introducir el nombre con el que desea hacer la reserva. Finalmente el sistema imprimirá un ticket que confirma que la reserva ha sido realizada con éxito.	
Condiciones de Ejecución: 1. Que el cliente se conecte al terminal TPR	

Entrada / Pasos de ejecución:

1. El terminal le pregunta al cliente en que restaurante, día y hora quiere efectuar la reserva.
2. El TPR con ayuda del Centro de Reservas comprueba si en el restaurante especificado por el cliente hay mesas libres para esa hora y día.
3. Si existen mesas libres, le envía al TPR el plano del restaurante con las mesas libres ubicadas en el plano.
4. El TPR reconstruye el plano con las mesas que hay libres. Para cada mesa dibujada en el plano se da información sobre el tipo de mesa (Fumador o No fumador y el número de comensales que pueden ocupar dicha mesa).
5. El usuario selecciona la mesa e indica el número de comensales que van a ocuparla.
6. El TPR se lo comunica a la central de reservas que a su vez confirma con el restaurante que todo está correcto.
7. Si todo está bien, el TPR pide al usuario que el nombre con el que desea realizar la reserva, y éste lo teclea.
8. El terminal se lo comunica al Centro de Reservas, el cual hace la reserva cambiando el estado de la mesa a "reservada". El TPR imprime el ticket correspondiente a la reserva indicando el día, la hora, la mesa y el nombre de la reserva.

Resultado Esperado: Reserva de una mesa en el restaurante

Evaluación de la Prueba:

3. Caso de prueba 3: Caso alternativo1: Hacer reserva_tpr

Caso de Prueba de Aceptación	
Código: 3	**Historia de Usuario (Nro. y Nombre): Caso alternativo 1:Hacer reserva_tpr**
Nombre: Caso alternativo 1: Hacer reserva_tpr	
Descripción: Si el cliente selecciona un restaurante en el que no hay mesas libres para la fecha y hora indicada el terminal se lo comunica dándole la posibilidad de solicitar	

sugerencias sobre restaurantes disponibles para el día y hora requerida. En el caso de que el usuario solicite dicha ayuda se le proporcionará una lista de restaurantes para que proceda a seleccionar uno, siguiendo con el procedimiento de reserva de mesa a través del terminal TPR.

Condiciones de Ejecución:

1. Que el cliente se conecte al terminal TPR
2. Que no haya mesas libres para la fecha y hora indicada en el restaurante elegido.
3. Que el cliente solicite ayuda al sistema para encontrar otro restaurante con mesas libres.

Entrada / Pasos de ejecución:

1. El terminal le pregunta al cliente en que restaurante, día y hora quiere efectuar la reserva.
2. El TPR con ayuda del Centro de Reservas comprueba si en el restaurante especificado por el cliente hay mesas libres para esa hora y día.
3. Si no hay mesas libres a la hora indicada por el usuario, el TPR pregunta al cliente si quiere sugerencias sobre restaurantes disponibles a la hora y día seleccionado.
4. En el caso en el que el cliente quiera sugerencias, el Centro de Reservas a través del TPR le facilita una lista de posibles restaurantes.
5. El usuario selecciona uno de los restaurantes de la lista y el centro de reservas le envía al TPR el plano del restaurante con las mesas libres ubicadas en el plano.
6. El TPR reconstruye el plano con las mesas que hay libres. Para cada mesa dibujada en el plano se da información sobre el tipo de mesa (Fumador o No fumador y el número de comensales que pueden ocupar dicha mesa).
7. El usuario selecciona la mesa e indica el número de comensales que van a ocuparla.
8. El TPR se lo comunica a la central de reservas que a su vez confirma con el restaurante que todo está correcto.
9. Si todo está bien, el TPR pide al usuario que el nombre con el que desea realizar la reserva, y este se lo teclea.
10. El terminal se lo comunica al Centro de Reservas, el cual hace la reserva cambiando el estado de la mesa a "reservada". El TPR imprime el ticket correspondiente a la reserva indicando el día, la hora, la mesa y el nombre de la reserva.

Resultado Esperado: Reserva de una mesa en el restaurante

Evaluación de la Prueba:

4. Caso de prueba 4: Caso alternativo2: Hacer reserva_tpr

Caso de Prueba de Aceptación	
Código: 4	**Historia de Usuario (Nro. y Nombre): Caso alternativo 2:Hacer reserva_tpr**
Nombre: Caso alternativo 2: Hacer reserva_tpr	
Descripción: Si el cliente a la hora de realizar la reserva no encuentra ninguna mesa que satisfaga sus necesidades podrá solicitar al sistema que le indique otro restaurante que tenga mesas libres para el día y hora seleccionada, pudiendo realizar la reserva de la mesa que elija de forma normal.	
Condiciones de Ejecución: 1. Que el cliente se conecte al terminal TPR 2. Que el cliente no encuentre ninguna mesa que satisfaga sus necesidades en el restaurante elegido. 3. Que el cliente solicite al sistema otro restaurante de la cadena.	
Entrada / Pasos de ejecución: 1. El terminal le pregunta al cliente en que restaurante, día y hora quiere efectuar la reserva. 2. El TPR con ayuda del Centro de Reservas comprueba si en el restaurante especificado por el cliente hay mesas libres para esa hora y día. 3. Si existen mesas libres, le envía al TPR el plano del restaurante con las mesas libres ubicadas en el plano. 4. El TPR reconstruye el plano con las mesas que hay libres. Para cada mesa dibujada en el plano se da información sobre el tipo de mesa (Fumador o No fumador y el número de comensales que pueden ocupar dicha mesa). 5. Si el cliente no encuentra ninguna mesa que le satisfaga solicita al sistema que le indique otro restaurante de la cadena que también tenga mesas libres a esa hora.	

6. El sistema le envía un listado de todos los restaurantes que cumplen sus condiciones.
7. El cliente selecciona el restaurante en el que desea realizar reserva.
8. El Centro de Reservas envía al TPR el plano del restaurante con las mesas libres ubicadas en el plano.
9. El TPR reconstruye el plano con las mesas que hay libres. Para cada mesa dibujada en el plano se da información sobre el tipo de mesa(Fumador o No fumador y el número de comensales que pueden ocupar dicha mesa).
10. El usuario selecciona la mesa e indica el número de comensales que van a ocuparla.
11. El TPR se lo comunica a la central de reservas que a su vez confirma con el restaurante que todo está correcto.
12. Si todo está bien, el TPR pide al usuario que el nombre con el que desea realizar la reserva, y este se lo teclea.
13. El terminal se lo comunica al Centro de Reservas, el cual hace la reserva cambiando el estado de la mesa a "reservada". El TPR imprime el ticket correspondiente a la reserva indicando el día, la hora, la mesa y el nombre de la reserva.

Resultado Esperado: Reserva de una mesa en el restaurante

Evaluación de la Prueba:

5. Caso de prueba 5: Activar reserva_tpr

Caso de Prueba de Aceptación	
Código: 5	**Historia de Usuario (Nro. y Nombre): Activar reserva_tpr**
Nombre: Activar reserva_tpr	
Descripción: El cliente al llegar al restaurante en el que se ha realizado la reserva deberá presentar el ticket entregado cuando esta se realizó. Si la hora a la que ha llegado no sobrepasa los 20 minutos de la hora reservada el cliente ocupara la mesa correspondiente.	
Condiciones de Ejecución: 1. Que el cliente haya hecho una reserva en el restaurante a una hora y día determinado.	

2. Que el cliente presente el ticket que se le entregó cuando realizó la reserva. 3. Que el cliente llegue al restaurante sin haber sobrepasado en 20 minutos la hora de reserva establecida.
Entrada / Pasos de ejecución: 1. El cliente que acaba de llegar al restaurante entrega el ticket de reserva. 2. Si la hora de llegada al restaurante no es superior a 20 minutos de la hora de reserva, el camarero cambia el estado de la mesa de "reservada" a "ocupada" y les sienta en el lugar correspondiente.
Resultado Esperado: Activación de la reserva de una mesa
Evaluación de la Prueba:

6. Caso de prueba 6: Caso alternativo 1: Activar reserva_tpr

Caso de Prueba de Aceptación	
Código: 6	**Historia de Usuario (Nro. y Nombre): Caso alternativo 1:Activar reserva_tpr**
Nombre: Caso alternativo 1: Activar reserva_tpr	
Descripción: Si la hora de llegada del cliente al restaurante sobrepasa los 20 minutos de la hora en la que se ha reservado el sistema habrá anulado la reserva, dejando la mesa libre para otro posible cliente.	
Condiciones de Ejecución: 1. Que el cliente haya hecho una reserva en el restaurante a una hora y día determinado. 2. Que el cliente presente el ticket que se le entregó cuando realizó la reserva. 3. Que el cliente llegue al restaurante habiendo sobrepasado en 20 minutos la hora de reserva establecida.	

Entrada / Pasos de ejecución: 1. Si el Centro de Reservas ha detectado que la hora de llegada del cliente al restaurante es superior a 20 minutos de la hora de la reserva, cambiará el estado de la mesa de "reservada" a "libre".
Resultado Esperado: Activación de la reserva de una mesa
Evaluación de la Prueba:

7. Caso de prueba 7: Hacer reserva_maitre

Caso de Prueba de Aceptación	
Código: 7	**Historia de Usuario (Nro. y Nombre): Hacer reserva_maitre**
Nombre: Caso alternativo 1: Hacer reserva_maitre	
Descripción: Cuando un cliente sin reserva o un cliente que tenia reserva, pero ha sido anulada por sobrepasar los 20 minutos de dicha reserva quiere una mesa, el maitre solicita al sistema un plano con las mesas libres, si existen, introduce los datos proporcionados por el cliente para una nueva reserva (número de comensales, fumador/no fumador…).	
Condiciones de Ejecución: 1. Que el cliente no haya realizado la reserva o que si la haya realizado pero la hora de llegada al restaurante haya sobrepasado 20 minutos de la hora de reserva, por lo que su mesa ha dejado de estar reservada. 2. Que el maître solicite al sistema el plano de mesas libres y existan en el restaurante.	

Entrada / Pasos de ejecución:

1. Cuando un cliente que no tiene reserva o que se ha anulado la que ya tenía, desea una mesa solicita al sistema que le muestre las mesas libres que existen en ese momento.
2. El encargado de las reservas le pregunta al usuario que tipo de mesa desea.
3. En caso que exista mesa libre con esas características el encargado hace la reserva y les sienta.
4. El responsable cambiará el estado de la mesa de libre a "ocupada".

Resultado Esperado: Reserva de una mesa en el restaurante

Evaluación de la Prueba:

8. Caso de prueba 8: Caso alternativo 1: Hacer reserva_maitre

Caso de Prueba de Aceptación	
Código: 8	**Historia de Usuario (Nro. y Nombre): Caso alternativo 1:Hacer reserva_maitre**
Nombre: Caso alternativo 1: Hacer reserva_maitre	
Descripción: Si la reserva ha sido anulada el maître comprueba a través del terminal si aún existen mesas libres. Si no es así pedirá al sistema el tiempo aproximado para que una quede libre. Si el cliente lo desea se deberá informar de otros restaurantes de la cadena que sí cuenten con mesas libres.	
Condiciones de Ejecución: 1. Que el cliente haya sobrepasado los 20 minutos de la hora de la reserva y su mesa	

haya quedado libre, por lo que habrá perdido su reserva. 2. Que el maître compruebe que no hay mesas libres en ese momento en el restaurante.
Entrada / Pasos de ejecución: 1. Cuando un cliente que no tiene reserva o que se ha anulado la que ya tenia desea una mesa solicita al sistema que le muestre las mesas libres que existen en ese momento. 2. El encargado de las reservas le pregunta al usuario que tipo de mesa desea. 3. En caso de que no exista mesa libre el encargado pide al sistema el tiempo aproximado para que quede libre la próxima mesa de las características de la mesa solicitada. 4. Si el cliente lo desea, se le informa de otros restaurantes de la cadena que sí tengan mesas libres.
Resultado Esperado: Reserva de una mesa en el restaurante
Evaluación de la Prueba:

Una vez realizados todos y cada uno de los casos de prueba habremos comprobado para cada historia de usuario que todo se lleva a cabo siguiendo los pasos correctamente y teniendo el resultado esperado. Si hubiéramos encontrado algún fallo se habría remediado de inmediato para evitar posibles errores en fases más tardías, en las cuales sería más difícil solucionarlos.

Con esto conseguimos un diseño limpio de errores para seguir con nuestro desarrollo, aunque estas pruebas se deberán seguir haciendo durante todo el proceso ya que estas deben ser continuas, frecuentemente repetidas y automatizadas. Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.

Como hemos mencionado antes, el estudio de este proyecto se basa en la toma de tiempos para demostrar que siguiendo unos patrones concretos, la aplicación de la metodología XP se realiza más fácilmente y con mayor rapidez. En esta fase del proyecto, estos tiempos se basan en la cantidad de tiempo empleado en la realización de las actividades anteriormente expuestas (historias de usuarios, estimación y casos de prueba) para un programador sin experiencia y sin seguir ningún método concreto de aplicación de la metodología.

Estos tiempos son los siguientes:

- Realización de historias de usuarios: 1 hora 45 minutos.
- Estimación de historias de usuarios: 45 minutos.
- Realización de casos de prueba: 2 horas 45 minutos.

Más adelante analizaremos estos datos con los demás tomados en siguientes fases.

Pasamos ahora a cumplimentar la segunda fase del proyecto. Esta se basa en la realización de patrones de producto los cuales utilizaremos más tarde.

12.4 Fase Segunda: Realización de Patrones

En esta segunda fase del proyecto hemos decidido realizar una serie de patrones de productos que más tarde aplicaremos a nuestro caso práctico del restaurante.

Con la utilización de estos patrones lo que queremos demostrar es que siguiendo unos pasos y doctrinas especificas y reutilizables es más fácil y rápida la aplicación de las metodologías Ágiles, en nuestro caso la metodología XP.

Los analistas/diseñadores con gran experiencia aplican, de forma mayormente intuitiva y automática, criterios precisos que, de forma global, solucionan de forma elegante y efectiva los problemas elegante y efectiva los problemas de modelado software de sistemas reales.

Usualmente estos diseñadores utilizan métodos, estructuras y subsistemas que son a la vez, herramientas de diseño y partes de la solución final.

Los ingenieros de software se enfrentan cada día a multitud de problemas de distinto calibre.

La "efectividad" de un ingeniero se mide por su rapidez y acierto en la diagnosis, identificación y resolución de tales problemas.

El mejor ingeniero es el que más utiliza la misma solución –matizada– para resolver problemas similares.

Debe existir alguna forma de comunicar al resto de los ingenieros los resultados encontrados tras mucho esfuerzo por alguno(s) de ellos. Se necesita, al fin, algún esquema de documentación que permita tal comunicación.

La mera documentación de líneas de código resulta insuficiente, pues únicamente fomenta el uso de la técnica "cut & plaste".

El tipo de los problemas y soluciones a documentar es muy variado:

- Programación
- Análisis
- Arquitectura
- Gestión
- Etc.

Se necesita un formato de documentación único que aúne conceptualmente estos distintos tipos.

"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces de la misma forma." Christopher Alexander

Si aceptamos que los patrones pueden resultar útiles en el desarrollo de software, el siguiente paso es reunirlos en catálogos de forma que resulten accesibles mediante distintos criterios, pues lo que necesitamos no es tan solo la completa descripción de cada uno de los patrones sino, esencialmente, la correspondencia entre un problema real y un patrón (o conjunto de patrones) determinado.

Lo que se pretende con un catálogo de patrones no es favorecer al diseñador experto (que quizás no necesite en absoluto de los patrones), sino más bien ayudar al diseñador inexperto a adquirir con cierta rapidez las habilidades de aquél, como también comunicar al posible cliente, si es el caso, las decisiones de diseño de forma clara y autosuficiente.

Un catálogo de patrones es un medio para comunicar la experiencia de forma efectiva, reduciendo lo que se conoce como "curva de aprendizaje" del diseño.

Tipos de patrones:

- De creación: Abstraen el proceso de creación de instancias.
- Estructurales: Se ocupan de cómo clase y objeto son utilizados para componer estructuras de mayor tamaño.
- De comportamiento: Atañen a los algoritmos y a la asignación de responsabilidad entre objetos.

Los patrones de software no son más que un conjunto de literatura sobre la resolución de problemas habituales en el diseño de software.

Esta literatura que recoge los patrones empezó teniendo una forma fija, de forma que se existía una especie de plantilla fija, con secciones tales como motivación, explicación, etc, que debía ser rellenada. Algunos autores han desarrollado sus propias plantillas o formato para los patrones, y otros simplemente optan por una descripción más relajada basada en textos sencillos sin orden preestablecido.

Los patrones de diseño pretenden:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Asímismo, no pretenden:

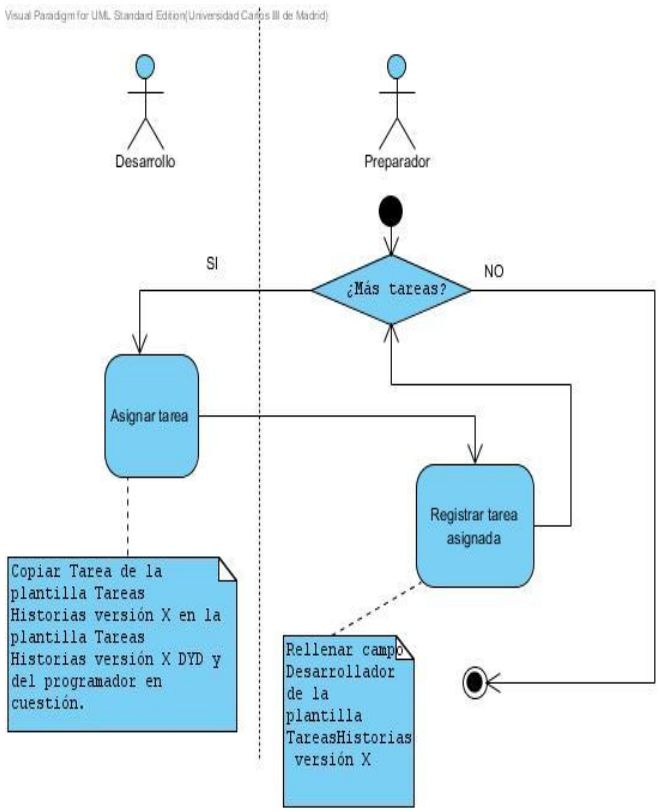
- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.

No es obligatorio utilizar los patrones, solo es aconsejable en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. Abusar o forzar el uso de los patrones puede ser un error.

En nuestro caso los patrones creados para la mejor aplicación de XP, ubicados en la URL http://kovachi.sel.inf.uc3m.es, son los siguientes:

12.4.1 Patrón Asignar Tarea

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Asignar Tarea
PATRONES RELACIONADOS	• Patrón Transformar Historia en Tareas • Patrón Estimar Tarea
CONTEXTO INICIAL	• Este producto puede usarse cuando deba distribuirse las tareas técnicas de implementación de requerimientos del área de negocio entre un grupo de trabajo de desarrolladores.
CONTEXTO RESULTATE	• El preparador del proyecto distribuye entre su equipo de trabajo las tareas técnicas a realizar para la implementación de los requerimientos del área de negocio, obteniendo así un plan de trabajo del equipo, con tareas a realizar y responsable de cada tarea.
PROBLEMA	• El preparador del proyecto debe ser capaz de repartir las tareas entre su equipo de manera equitativa y teniendo en cuenta las capacidades, habilidades, predisposiciones y posibles reacciones de cada uno de los miembros de su equipo.
RESTRICCIONES (FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de Cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
	Tiempo de Desarrollo: • Para crear el Patrón de Producto: 45 minutos.

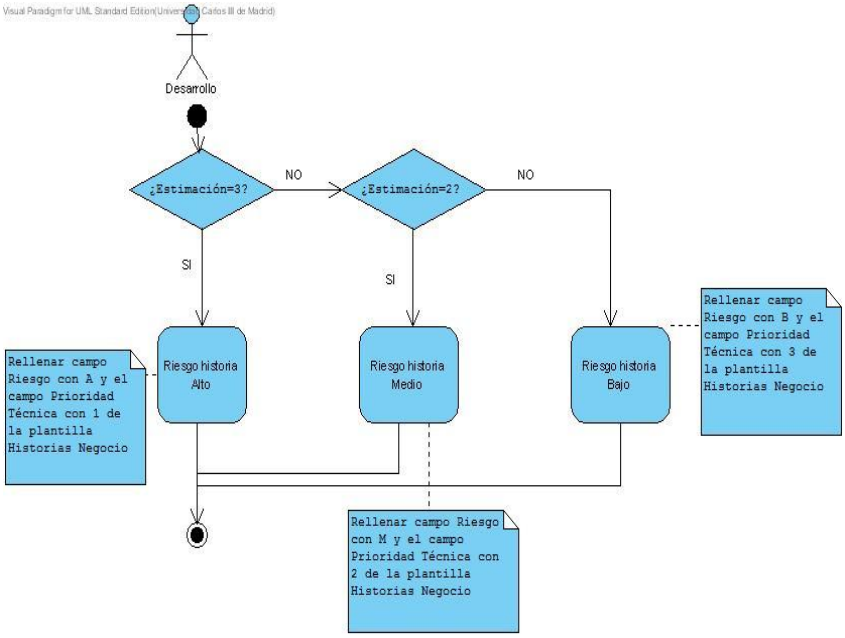
SOLUCIÓN	**Solución:** 
ROLES	• Desarrolladores (2 a 12) • Preparador (1)
ENTRADAS	• Tareas Historias versión X
LECCIONES APRENDIDAS	• Deben repartirse las tareas en función de la complejidad de las mismas, la carga de trabajo de cada miembro del equipo y por supuesto de si las habilidades del miembro le van a permitir realizar con mayor o menor facilidad la tarea encomendada. • El preparador reparte las tareas a implementar entre sus colaboradores según las fichas de valoración de cada uno de los miembros de su equipo. • En el documento de salida *Tareas Historias versión X*, el campo responsable de realización, de cada tarea, ha de estar relleno.

PLANTILLAS	• Tareas_Historias_Ver_X.doc • Tareas_Historias_Ver_X_DyD_Y.doc
EJEMPLOS	No hay ejemplos sobre el uso de este Patrón de Producto en este momento.
SALIDAS	• Tareas Historias versión X • Tareas Historias versión X Programador Y
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. • El miembro que asigna las tareas será necesario que cuente con el conocimiento de las habilidades de cada uno de los miembros del equipo para realizar un buen reparto en función de éstas habilidades y de la complejidad de dichas actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for

	UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A proposito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming.Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica.Ed Addisson Wesley.

12.4.2 Patrón Clasificar por Riesgo:

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Clasificar por Riesgo
PATRONES RELACIONADOS	• Patrón Estimar Historia • Patrón Dividir Historia • Patrón Establecer Velocidad
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto que requiera una priorización por parte del área de tecnología de las historias escritas por el área de negocio y posteriormente estimadas por el propio área de tecnología. Esta priorización esta marcada por el riesgo de que la nueva funcionalidad descrita por las historias no sea entregada al área de negocio en el plazo de tiempo estimado. Dentro de esta clasificación se establecen tres grupos, las historias cuyo riesgo sea elevado, medio o bajo (estimación de 3, 2 y 1 semanas respectivamente). Se entiende que las historias cuya estimación es mayor es debido a que la complejidad en su implementación es mayor.
CONTEXTO RESULTATE	Se obtendrá el conjunto de historias estimadas por tecnología pero ordenadas según el riesgo de no ser entregadas al área de negocio en el plazo de tiempo estimado. Dentro del ese conjunto se tendrán 3 subconjuntos (riesgo alto, medio, bajo).
PROBLEMA	El área de tecnología debe haber estimado las historias en función a la complejidad que tiene la implementación de cada una de ellas. A mayor complejidad mayor es la estimación, con un tope de 3 semanas. Se establecerá como historia de riesgo elevado a las de mayor estimación (3 semanas), riesgo medio a las historias estimadas en 2 semanas y riesgo bajo a las estimadas en 1 semana.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de

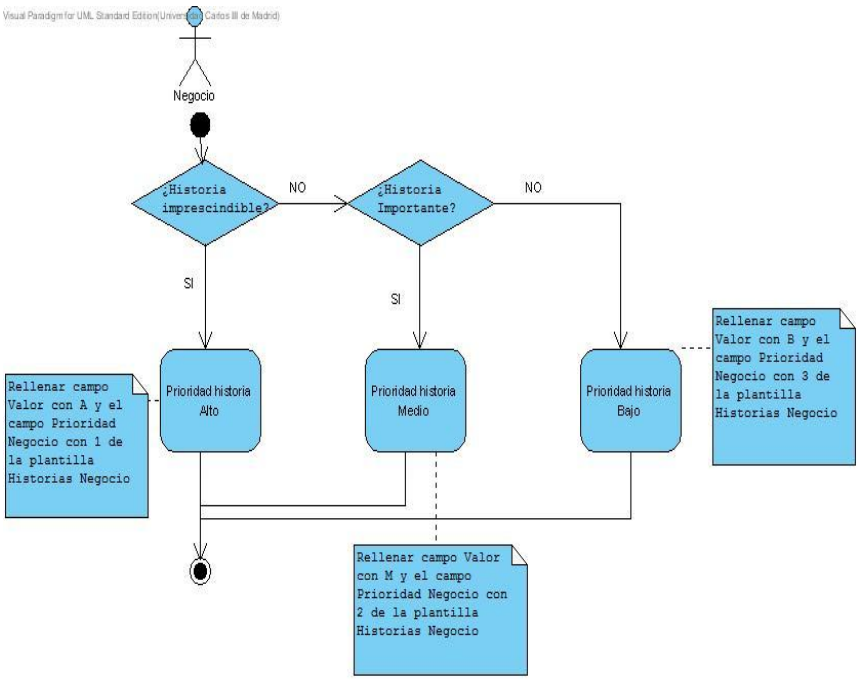
	usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si la estimación de la tarea es 3 se establece riesgo alto, si es 2 el riesgo es medio, si es 1 el riesgo es bajo.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 15 minutos. **Solución:** 
ROLES	• Desarrolladores (2 a 12) • Director de proyecto de desarrollo (1) • Preparador (1) • Controlador (1) Nota: El preparador y el controlador pueden ser la misma persona.

ENTRADAS	• Historias de Negocio
LECCIONES APRENDIDAS	• Las historias con una estimación en duración mayor son las que a priori parecen más complejas en su desarrollo y por tanto en las que pueden surgir mayores complicaciones. Por esto deben ser clasificadas por encima del resto. • Hay que rellenar el campo indicando el riesgo de implementar a tiempo la historia desde el punto de vista de tecnología. Existen tres categorías (A-Riesgo Alto, M-Riesgo Medio, B-Riesgo Bajo). Se actualiza la plantilla con la estimación de tecnología. Se registra la velocidad.
PLANTILLAS	• Historias_Negocio.doc
EJEMPLOS	No hay ejemplos sobre el uso de este Patrón de Producto en este momento.
SALIDAS	• Historias de Negocio
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el

	conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UMLpara la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming.Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A proposito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming.Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica.Ed Addisson Wesley.

12.4.3 Patrón Clasificar por Valor:

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Clasificar por Valor
PATRONES RELACIONADOS	• Patrón Estimar Historia • Patrón Dividir Historia • Patrón Establecer Velocidad
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto que requiera una priorización por parte del área de negocio de las historias escritas por ellos mismos y posteriormente estimadas por el área de tecnología. Esta priorización esta marcada por el valor que la nueva funcionalidad aportara al negocio cuando sea desarrollada. Dentro de esta clasificación se establecen tres grupos, las historias que describen funcionalidades imprescindibles, las importantes y las misceláneas.
CONTEXTO RESULTATE	Se obtendrá el conjunto de historias estimadas por tecnología pero ordenadas según el valor aportado al negocio. Dentro del ese conjunto se tendrán 3 subconjuntos (imprescindibles, importantes y miscelánea).
PROBLEMA	El área de negocio debe tener muy clara la función que representa cada historia en el aplicativo a desarrollar y no dejarse llevar por las historias que representan funcionalidades irrelevantes (miscelánea) en vez de las imprescindibles, por ejemplo.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** El usuario debe haber realizado estudios formales de viabilidad funcional con los pros y contras de las funcionalidades descritas por cada una de las historias para poder lanzar una opinión fiable. Lo

	primero en realizar debe ser lo imprescindible.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 15 minutos. **Solución:** 
ROLES	• Usuarios del área de negocio (2 como mucho)
ENTRADAS	• Historias de Negocio
LECCIONES APRENDIDAS	• La visión del aplicativo que se desea obtener debe ser clara. Las funcionalidades más importantes deben clasificarse por encima de las que son meramente decorativas. • Hay que rellenar el campo indicando la prioridad de la historia desde el punto de vista de negocio. Existen tres categorías (A-(alto) Imprescindible, M-(medio)Necesario, B-(bajo)Miscelánea).
PLANTILLAS	• Historias_Negocio.doc
EJEMPLOS	No hay ejemplos sobre el uso de este Patrón de Producto en este momento.

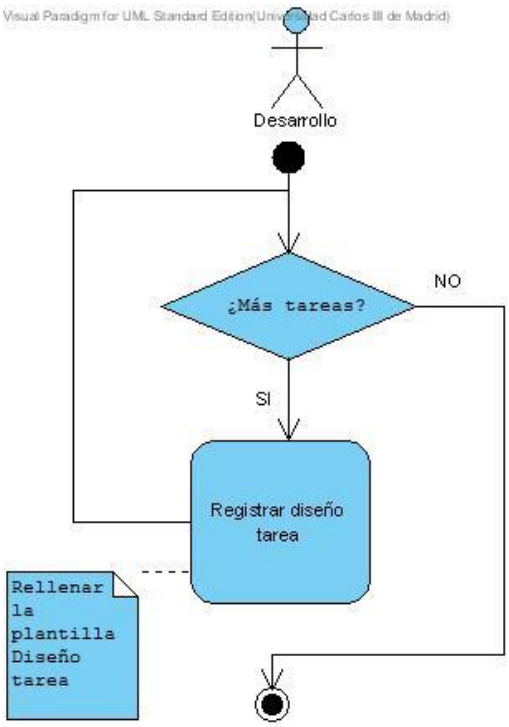
SALIDAS	• Historias de Negocio
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación

RECURSOS DE INFORMACIÓN	extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica.Ed Addisson Wesley.

12.4. 4 Patrón Diseñar

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Diseñar
PATRONES RELACIONADOS	• Patrón Pruebas de Integración • Patrón Modificar Integración • Patrón Generar Código
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que se siga la filosofía de trabajo en parejas para la implementación de código. El diseño crea una estructura que organiza la lógica del sistema,

	un buen diseño permite que el sistema crezca con cambios en un solo lugar.
CONTEXTO RESULTATE	Se obtienen diseños sencillos de pequeñas partes del sistema.
PROBLEMA	El diseño en parejas no consiste en que una persona escriba y la otra mire, es un dialogo entre dos personas que intentan simultáneamente programar, diseñar, analizar y probar. Debe elegirse una estrategia de diseño sencilla que de lugar a un diseño sencillo, además debe encontrarse rápidamente la manera de comprobar la calidad del diseño elaborado. Puede surgir incertidumbre (¿Cuándo añadimos más a lo diseñado?), por ejemplo: 1. Algunas veces el cliente elimina la característica del diseño por anticipado. 2. A veces se aprende una manera mejor de trabajar cuando se está trabajando de otra manera. Los diseños deben de ser sencillos, si alguna parte del sistema es de desarrollo complejo, lo apropiado es dividirla en varias. Si hay fallos en el diseño o malos diseños, estos deben ser corregidos cuanto antes.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente**: Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
	Tiempo de Desarrollo: • Para crear el Patrón de Producto: 45 minutos.

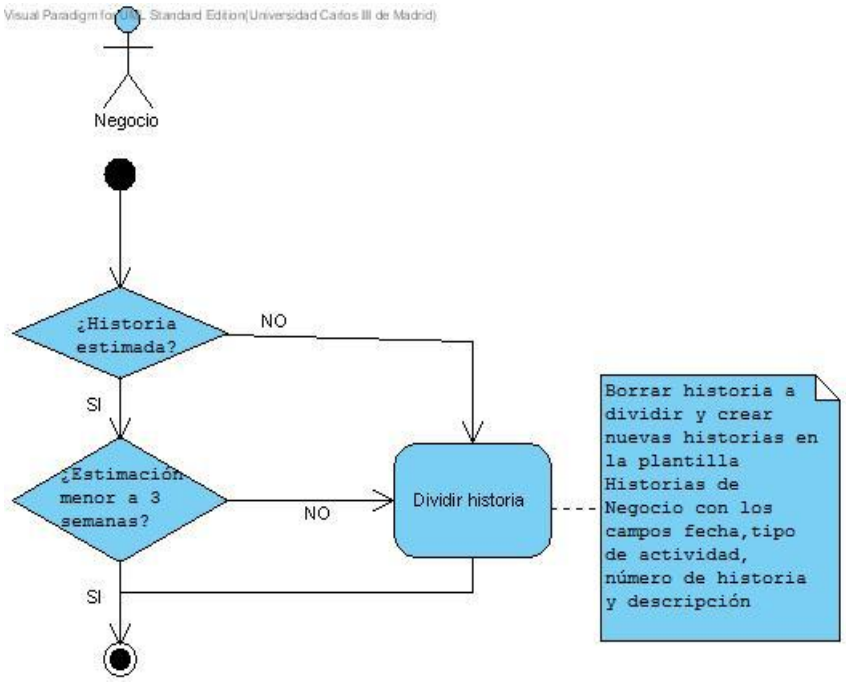
SOLUCIÓN	**Solución:** 
ROLES	• Desarrolladores (2 a 12)
ENTRADAS	• Pruebas
LECCIONES APRENDIDAS	• No hay que ser ambiciosos inicialmente. No debemos esperar un beneficio muy alto inicialmente. Hay que ser pacientes e ir añadiendo al diseño según avanza el desarrollo. La estrategia de diseño debe ser: 1. Comenzar con una prueba, así sabremos lo que estamos haciendo. 2. Diseñar e implementar justo lo suficiente para que la prueba funcione. 3. Repetir. 4. Si siempre se ve la posibilidad de hacer el diseño más simple, hacerlo. • Diseñar inicialmente lo más sencillo posible para ir incrementando su valor. • El archivo *Pruebas* es un documento genérico, para todos los tipos de prueba con la casilla pruebas unitarias marcada, que aglutina todos los casos de prueba para una

	tarea en cuestión. En este documento se refleja su responsable e historia asociada. • El archivo *Diseño Tarea* es un documento con el diseño de la tarea a implementar.
PLANTILLAS	• Diseño_Tarea.doc
EJEMPLOS	No hay ejemplos sobre el uso de este Patrón de Producto en este momento.
SALIDAS	• Diseño Tarea
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for

	UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.5 Patrón Dividir Historia

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Dividir Historia
PATRONES RELACIONADOS	• Patrón Estimar Historia • Patrón Clasificar por Valor • Patrón Clasificar por Riesgo
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en los que una vez escrita la historia y entregada al área de desarrollo, estos no pueden estimarla debido a que o es muy compleja para ser tratada unitariamente, con lo que debe dividirse en otras más pequeñas. Las historias de usuarios deben poder ser programadas en un tiempo entre una y tres semanas. Si la estimación es superior a tres semanas, debe ser dividida en dos o más historias. Si es menos de una semana, se debe combinar con otra historia.
CONTEXTO RESULTATE	Se obtendrá un nuevo conjunto de historias (2 a n) por cada historia inicial que no pudo ser estimada por el área de tecnología.
PROBLEMA	El área de negocio debe ser capaz de dividir las historias no estimadas en un número de historias determinadas que puedan ser tratadas unitariamente. Deben ser lo más sencillas posibles.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar**: Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el

	área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso**: Se deben establecer reuniones entre las áreas tecnológicas y de negocio donde la comunicación sea fluida y haya confianza mutua. El espacio de trabajo debe estar organizado conforme a las directrices de la metodología. Es importante que no falte comida y bebida para que las reuniones sean distendidas.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos. **Solución:** 
ROLES	• Usuarios del área de negocio (2 como mucho)
ENTRADAS	• Historias de Negocio
LECCIONES APRENDIDAS	• Las historias deben ser lo más atómicas posibles, de no serlo serán devueltas por el área de tecnología. Una funcionalidad una historia. • Establecer reuniones en las que siempre participe el área

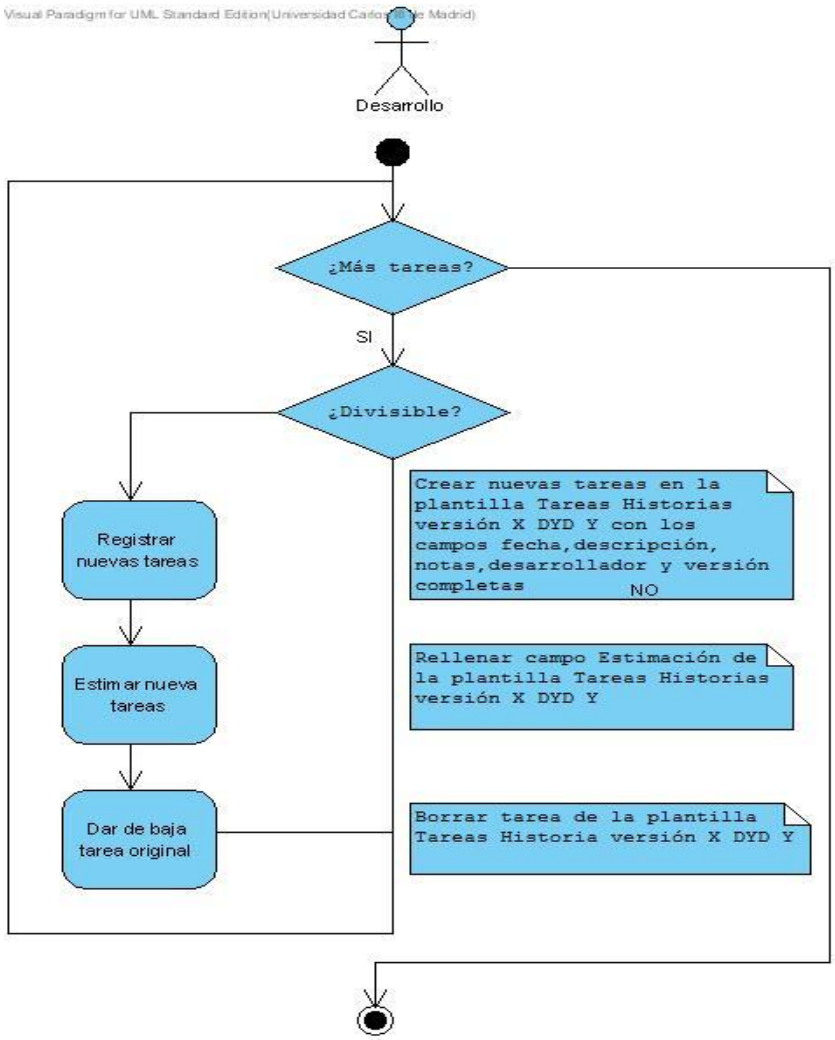
	de negocio junto con el equipo de desarrollo (programadores incluidos).
PLANTILLAS	• Historias_Negocio.doc
EJEMPLOS	No hay ejemplos sobre el uso de este Patrón de Producto en este momento.
SALIDAS	• Historias de Negocio
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.

RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.6 Patrón Dividir Tarea

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Dividir Tarea
PATRONES RELACIONADOS	• Patrón Estimar Tarea
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que debido al resultado de la estimación de una tarea, por el cual no pueda completarse en tiempo, esta deba dividirse en varias más sencillas para permitir su finalización a tiempo.

CONTEXTO RESULTATE	Los desarrolladores obtienen nuevas tareas como resultado de la separación de otras más costosas en tiempo.
PROBLEMA	Los desarrolladores deben ser capaces de estimar cada tarea, y de dividir las tareas que necesiten más de unos días en ser implementadas o bien agruparlas.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar**: Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos.

	Solución: 
ROLES	• Desarrolladores (2 a 12)
ENTRADAS	• Tareas Historias versión X • Tareas Historias versión X Programador Y

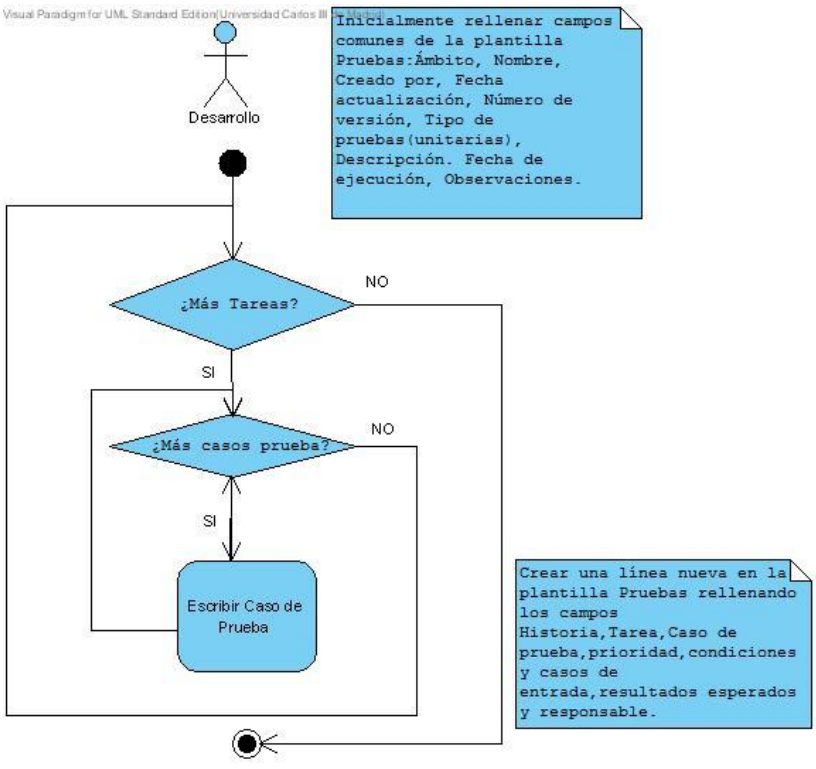
LECCIONES APRENDIDAS	• El programador debe dividir una tarea cuya estimación supere un cierto número de días. • Los desarrolladores dividen las tareas si la estimación de alguna de ellas supera unos pocos días. • En el documento de salida *Tareas Historias versión X* se añaden al final del documento las nuevas tareas resultantes de la división de las que tenían el campo en cuestión marcado, eliminando estas últimas tareas. • En el documento de salida *Tareas Historias versión X Programador Y* se añaden al final del documento las nuevas tareas resultantes de la división, eliminando estas últimas.
PLANTILLAS	• Tareas_Historias_Ver_X.doc • Tareas_Historias_Ver_X_DyD_Y.doc
EJEMPLOS	No hay ejemplos sobre el uso de este Patrón de Producto en este momento.
SALIDAS	• Historias de Negocio
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades.

	Habilidades: • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UMLpara la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La

	programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.7 Patrón Escribir casos de Prueba

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Escribir Casos de Prueba
PATRONES RELACIONADOS	• Patrón Establecer Balanceo • Patrón Pruebas Unitarias
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el se deban identificar casos de prueba para una funcionalidad concreta antes de su implementación. Las características del software que no pueden ser demostradas mediante pruebas, simplemente, no existen. Las pruebas dan la oportunidad de saber si lo implementado es lo que en realidad se tenía en mente. Las pruebas nos indican que nuestro trabajo funciona, cuando no podemos pensar en ninguna prueba que pudiese originar un fallo en nuestro sistema, entonces habremos acabado por completo.
CONTEXTO RESULTATE	Los desarrolladores obtienen un conjunto de casos de prueba relacionados con una funcionalidad en concreto a desarrollar.
PROBLEMA	Los desarrolladores deben escribir pruebas que ayuden a obtener un programa que funcione y que se mantenga funcionando a lo largo del tiempo. Los programadores escriben pruebas método a método bajo las siguientes circunstancias: 1. Si la interfaz para un método no esta clara, escribe una prueba antes de escribir el método. 2. Si la interfaz esta clara, pero imagina que la implementación podría ser un poco menos complicada, escribe una prueba antes de escribir el método. 3. Si piensa en una circunstancia no usual en la cual el código debiera funcionar conforme esta escrito, escribe una prueba para comunicar la circunstancia.

RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos. **Solución:** 

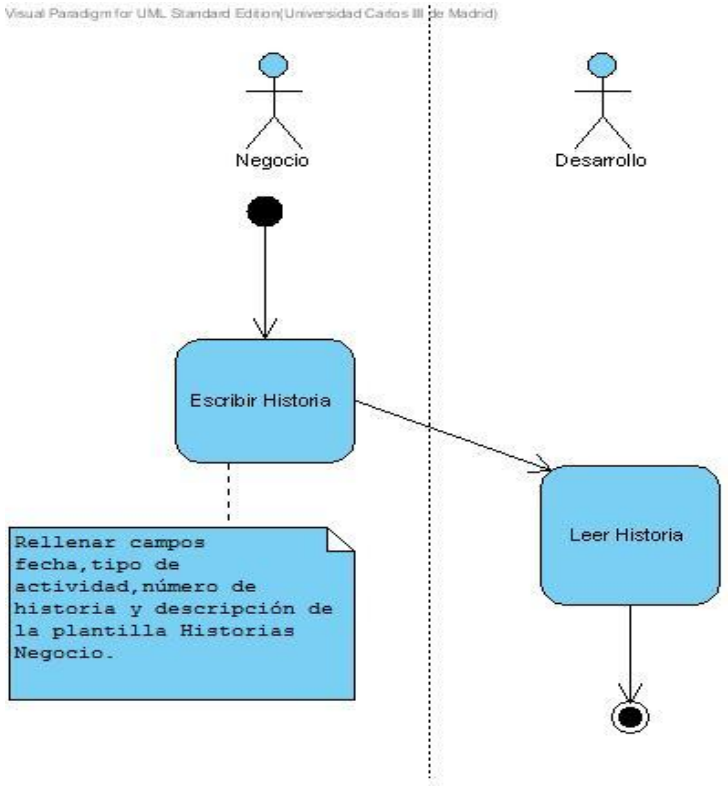
ROLES	• Desarrolladores (2 a 12)
ENTRADAS	• Tareas Historias versión X Programador Y
LECCIONES APRENDIDAS	• No se debe caer en la tentación de subestimar las pruebas. Trabajar en parejas de programadores puede reducir la posibilidad anterior. Las pruebas deben ser aisladas y automatizadas. El resultado de las pruebas debe ser positivo o negativo sin puntos intermedios. Como es imposible probar todo hay que escribir los casos de pruebas de lo que se piensa que puede fallar. Hay que tener especial cuidado con las pruebas que funcionan cuando no lo esperábamos (debemos revisar ese caso de prueba). Por tanto para escribir los casos de prueba debemos reflexionar sobre las pruebas que merece la pena hacer y cuáles no. • El archivo de salida *Pruebas.xls* es un documento genérico, para todos los tipos de prueba con la casilla pruebas unitarias marcada, que aglutina todos los casos de prueba para una tarea en cuestión. • Un desarrollador elige una tarea asignada y junto con un compañero analizan la funcionalidad.
PLANTILLAS	• Pruebas.xls
EJEMPLOS	Caso de Prueba de Aceptación Código: 1 | Historia de Usuario (Nro. y Nombre): Hacer reserva_tpr Nombre: Hacer reserva_tpr **Descripción:** El cliente se conecta e introduce en el sistema el nombre del restaurante, el dia y la hora a la que desea hacer la reserva. El restaurante con los datos recibidos comprueba la disponibilidad de mesas construyendo un plano del restaurante, ubicando las mesas libres. El cliente selecciona la mesa que mas se adecue a sus preferencias e indica el numero de comensales que la van a ocupar. Además el cliente deberá introducir el nombre con el que desea hacer la reserva. Finalmente el sistema imprimirá un ticket que confirma que la reserva ha sido realizada con éxito. **Condiciones de Ejecución:** 1. Que el cliente se conecte al terminal TPR **Entrada / Pasos de ejecución:** 1. El terminal le pregunta al cliente en que restaurante, día y hora quiere efectuar la reserva. 2. El TPR con ayuda del Centro de Reservas comprueba si en el restaurante especificado por el cliente hay mesas libres para esa hora y día. 3. Si existen mesas libres, le envía al TPR el plano del restaurante con las mesas libres ubicadas en el plano. 4. El TPR reconstruye el plano con las mesas que hay libres. Para cada mesa dibujada en el plano se da información sobre el tipo de mesa (Fumador o No fumador y el número de comensales que pueden ocupar dicha mesa). 5. El usuario selecciona la mesa e indica el número de comensales que van a ocuparla. 6. El TPR se lo comunica a la central de reservas que a su vez confirma con el restaurante que todo está correcto. 7. Si todo está bien, el TPR pide al usuario que el nombre con el que desea realizar la reserva, y éste lo teclea. 8. El terminal se lo comunica al Centro de Reservas, el cual hace la reserva cambiando el estado de la mesa a "reservada". El TPR imprime el ticket correspondiente a la reserva indicando el día, la hora, la mesa y el nombre de la reserva. **Resultado Esperado:** Reserva de una mesa en el restaurante **Evaluación de la Prueba:**

SALIDAS	• Pruebas.xls
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Será necesario un editor de hojas de cálculo como OpenOffice Calc o Microsoft Excel. • Además se necesitará una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-

RECURSOS DE INFORMACIÓN	02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.8 Patrón Escribir Historia

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Escribir Historia
PATRONES RELACIONADOS	• Patrón Estimar Historia
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que el proceso de recogida de los requerimientos funcionales del negocio se realice recibiendo historias escritas, lo mas atómicas posibles, por parte del negocio. Además, se usa en los proyectos en los que los requerimientos son cambiantes y es preciso entregar el resultado del proyecto en una fecha determinada que implica un alto riesgo para la consecución del mismo.
CONTEXTO RESULTATE	Se obtendrán un conjunto de historias, escritas por el negocio, en las cuales describirán, de la manera más atómicas posibles, las funcionalidades del aplicativo a desarrollar. Las historias de usuarios representan una breve descripción del comportamiento del sistema, emplea terminología del cliente, sin lenguaje técnico, se realiza una por cada característica principal del sistema. Se emplean para hacer estimaciones de tiempo y para el plan de lanzamientos, reemplazan un gran documento de requisitos y presiden la creación de las pruebas funcionales.
PROBLEMA	El área tecnológica debe ser capaz de conseguir que el negocio entregue las historias lo mas sencillas y atómicas posibles.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo • **Heurísticas de uso:** Se deben establecer reuniones entre las áreas tecnológicas y de negocio donde la comunicación sea fluida y haya confianza mutua. El espacio de trabajo debe estar organizado conforme a las

	directrices de la metodología. Es importante que no falte comida y bebida para que las reuniones sean distendidas.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 15 minutos. **Solución:** 
ROLES	• Usuarios del área de negocio (2 como mucho) • Desarrolladores (2 a 12) • Director de proyecto de desarrollo (1) • Preparador (1) • Controlador (1)
ENTRADAS	• Historias de Negocio
LECCIONES APRENDIDAS	• Las reuniones deben de ser cortas y deben de estar dirigidas por un preparador del área de tecnología. Debe hacerse amena la reunión con comida, bebida, anécdotas,

<table>
<tr><td></td><td>etc…. El usuario de negocio debe estar enteramente dedicado al proyecto.

• Establecer reuniones en las que siempre participe el área de negocio junto con el equipo de desarrollo (programadores incluidos).</td></tr>
<tr><td>PLANTILLAS</td><td>• Historias_Negocio.doc</td></tr>
<tr><td>EJEMPLOS</td><td>
<table>
<tr><td colspan="2">Historia de Usuario</td></tr>
<tr><td>Número: 1</td><td>Nombre: Hacer reserva_TPR</td></tr>
<tr><td colspan="2">Usuario: CLIENTE</td></tr>
<tr><td>Modificación de Historia Número:</td><td>Iteración Asignada:</td></tr>
<tr><td>Prioridad en Negocio: Alto
(Alta / Media / Baja)</td><td>Puntos Estimados: 5</td></tr>
<tr><td>Riesgo en Desarrollo: Alto
(Alto / Medio / Bajo)</td><td>Puntos Reales:</td></tr>
<tr><td colspan="2">Descripción:
El cliente se conecta e introduce en el sistema el nombre del restaurante, el dia y la hora a la que desea hacer la reserva. El restaurante con los datos recibidos comprueba la disponibilidad de mesas construyendo un plano del restaurante, ubicando las mesas libres. El cliente selecciona la mesa que mas se adecue a sus preferencias e indica el numero de comensales que la van a ocupar. Además el cliente deberá introducir el nombre con el que desea hacer la reserva. Finalmente el sistema imprimirá un ticket que confirma que la reserva ha sido realizada con éxito.</td></tr>
<tr><td colspan="2">Observaciones:</td></tr>
</table>
</td></tr>
<tr><td>SALIDAS</td><td>• Historias de Negocio</td></tr>
<tr><td>CONTROLADORES DE CALIDAD</td><td>• Ninguno</td></tr>
<tr><td>NIVEL DE MADUREZ</td><td>Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).</td></tr>
<tr><td>CONOCIMIENTOS BÁSICOS</td><td>Conocimientos:

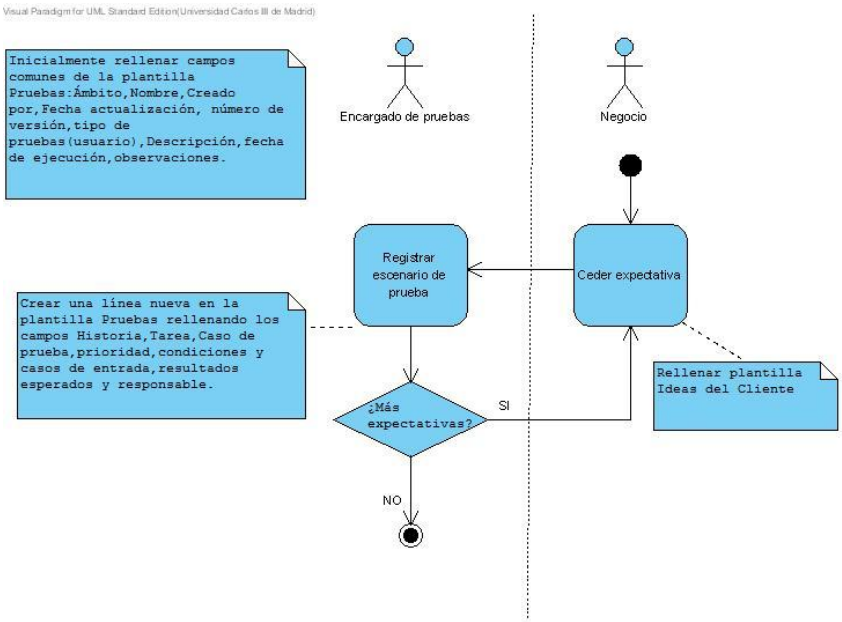
• Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona.
• Conocimiento de la visión común de cómo funciona el</td></tr>
</table>

	programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf

	• Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.9 Patrón Escribir Pruebas Funcionales

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Escribir Pruebas Funcionales
PATRONES RELACIONADOS	• Patrón Registrar Progreso • Patrón Recuperación Tareas • Patrón Pruebas Funcionales Historia
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que se deban realizar pruebas sobre escenarios descritos por el cliente.
CONTEXTO RESULTATE	El equipo de trabajo, clientes incluidos, obtienen un conjunto de escenarios sobre los que probar el código escrito.
PROBLEMA	El encargado de pruebas del proyecto, debe ser capaz de traducir las ideas del cliente a pruebas reales del ámbito técnico. Además, debe utilizar dichas ideas del cliente como punto de partida para las variaciones que probablemente harán que falle el software generado.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.

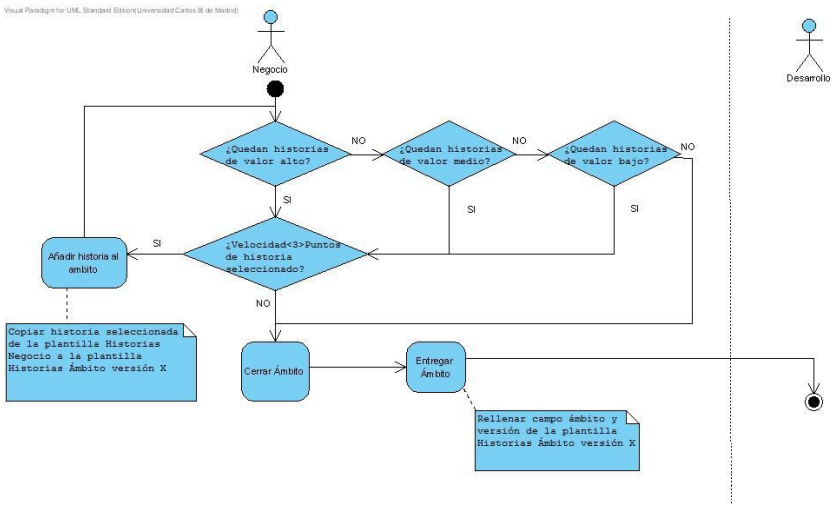
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos. **Solución:** 
ROLES	• Encargado de pruebas (1) • Usuarios del área de negocio (2 como mucho)
ENTRADAS	• Ideas Cliente
LECCIONES APRENDIDAS	• Los clientes escriben pruebas historia a historia. Se deben preguntar que tendrían que probar antes de estar seguros de que la historia en cuestión se ha completado. Los clientes necesitan alguna herramienta que les permita escribir, ejecutar, revisar y mantener las pruebas funcionales, es decir, una especie de traductor entre el lenguaje del cliente y de la parte técnica. • El encargado de pruebas del proyecto se reúne con los clientes y transforma sus ideas y expectativas sobre el software en escenarios (casos de prueba) reales que pueden realizarse dentro del ámbito tecnológico. • El documento de entrada *Ideas Cliente* contiene las ideas y expectativas del cliente de las funcionalidades que

	espera que realice el software. • El documento de salida *Pruebas* es un archivo genérico de tipo .xls (Microsoft Excel), para todos los tipos de prueba con la casilla pruebas usuario marcada, que recoge la traducción de las ideas del cliente, en el lenguaje del ámbito técnico, a escenarios de prueba del software generado.
PLANTILLAS	• Ideas_Cliente.doc • Pruebas.xls
EJEMPLOS	• Proyecto C3 de Daimler-Chrysler
SALIDAS	• Pruebas
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.

HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Será necesario un editor de hojas de cálculo como OpenOffice Calc o Microsoft Excel. • Además se necesitará una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.10 Patrón Establecer Ámbito Versión

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Establecer Ámbito Versión
PATRONES RELACIONADOS	• Patrón Establecer Velocidad • Patrón Transformar Historia en Tareas
CONTEXTO INICIAL	Este producto puede usarse en todos los proyectos en los que la entrega de aplicativo al área de negocio este organizado en versiones o de manera incremental.
CONTEXTO RESULTATE	El área de negocio obtiene un producto totalmente operativo y listo para funcionar con una funcionalidad determinada.
PROBLEMA	El área de negocio debe establecer el ámbito de la versión en cuestión. Son ellos los que determinan las historias que formaran parte de una versión determinada. El área de tecnología se limita a recibir la lista de historias seleccionada por el área de negocio.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos.

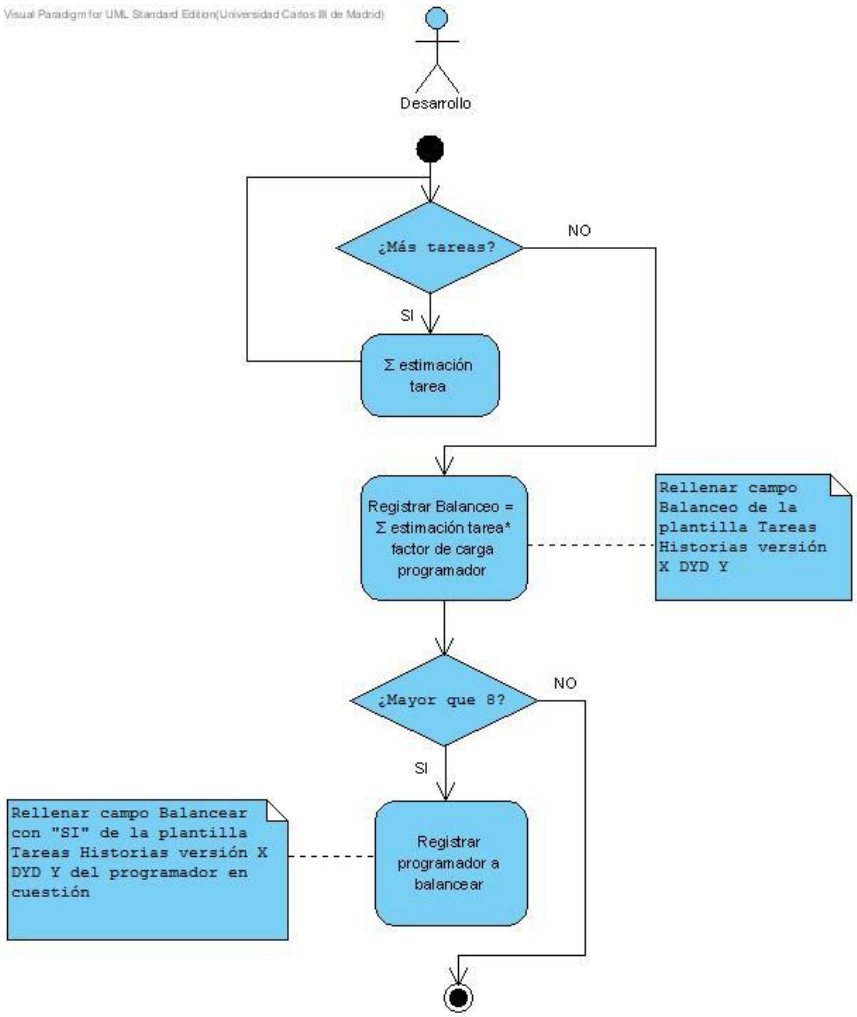
	Solución: 
ROLES	• Usuarios del área de negocio (2 como mucho) • Desarrolladores (2 a 12) • Director de proyecto de desarrollo (1) • Preparador (1) • Controlador (1)
ENTRADAS	• Historias de Negocio
LECCIONES APRENDIDAS	• Deberán elegirse las historias con el valor más alto para el negocio de entre las que queden sin implementar teniendo en cuenta las restricciones de velocidad del área de tecnología. Si se ajusta demasiado el numero de historias seleccionadas para cada ámbito a la velocidad de desarrollo se pueden producir retrasos en las entregas por sobrecarga de trabajo. • Negocio escoge para la versión en cuestión las historias que clasificó con mayor valor teniendo en cuenta la velocidad establecida por tecnología. El número de puntos de historia seleccionado para la versión no puede ser superior a la velocidad por 3. • En referencia al producto de salida Historias de Negocio, las historias seleccionadas se marcan con la versión

	correspondiente para llevar un control sobre ellas en el documento general de historias. • El producto de salida Historias Ámbito Versión X es una selección de historias para el ámbito de la versión correspondiente.
PLANTILLAS	• Historias_Ambito_Ver_X.doc
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Historias Ámbito Versión X • Historias de Negocio
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.

HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.11 Patrón Establecer Balanceo

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Establecer Balanceo
PATRONES RELACIONADOS	• Patrón Establecer Factor de Carga • Patrón Escribir Casos de Prueba
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que se haya observado que hay programadores con un factor de carga demasiado elevado.
CONTEXTO RESULTATE	Los desarrolladores obtienen ellos mismo la valoración de lo ocupados que están en el desarrollo del proyecto.
PROBLEMA	Los desarrolladores deben ser capaces de saber si están demasiado ocupados o por el contrario si están demasiado ociosos.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos.

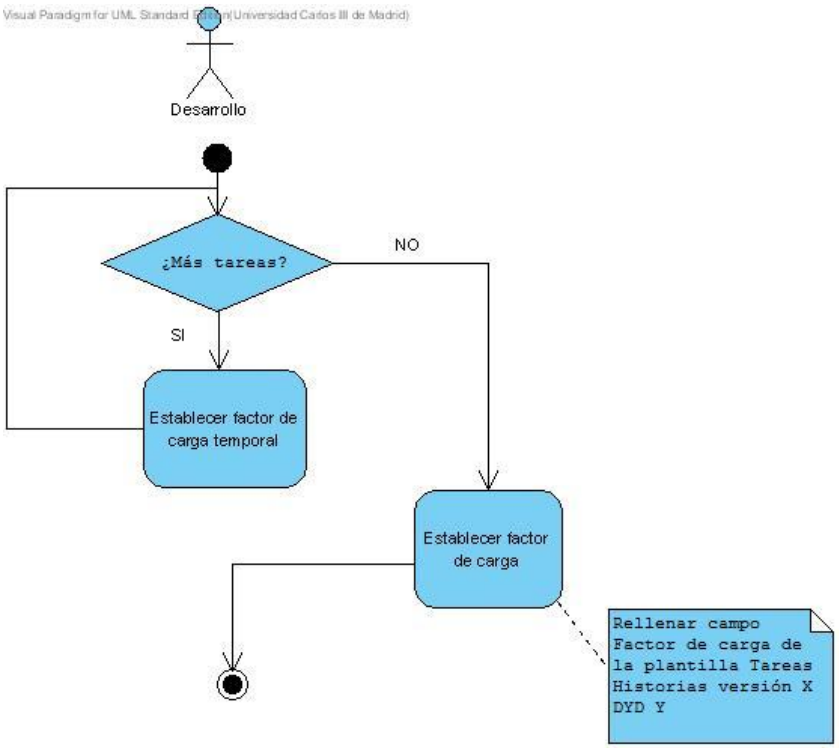
	Solución: 
ROLES	• Desarrolladores (2 a 12)
ENTRADAS	• Tareas Historias versión X Programador Y
LECCIONES APRENDIDAS	• Los programadores que se comprometen demasiado (el resultado de sumar las estimaciones de las tareas que les fueron encomendadas por su factor de carga es elevado con respecto al de los demás miembros del equipo) deben ser liberados de algunas tareas para establecer el equilibrio en el equipo de trabajo. Si todo el equipo esta demasiado comprometido debe encontrarse la forma de

	volver al equilibrio. Todos los programadores deben tener tiempo para realizar sus tareas y para ayudar a sus compañeros a realizar las suyas. • Los desarrolladores deben sumar las estimaciones de todas sus tareas y multiplicarlas por su factor de carga. • En el documento de salida *Tareas Historias versión X Programador Y* se rellena la cabecera del documento con el balanceo establecido por el programador en cuestión.
PLANTILLAS	• Tareas_Historias_Ver_X_DyD_Y.doc
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Tareas Historias versión X Programador Y
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.

HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A proposito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.12 Establecer Factor de Carga

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Establecer Factor de Carga
PATRONES RELACIONADOS	• Patrón Estimar Tarea • Patrón Dividir Tarea • Patrón Unir Tarea • Patrón Establecer Balanceo
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que los programadores deban establecer una medida para conocer su carga de trabajo. El factor de carga es el periodo de días para completar una tarea dividido por el tiempo ideal que el desarrollador ha estimado.
CONTEXTO RESULTATE	Los desarrolladores obtienen un factor de carga que les va a servir como medida de su carga de trabajo.
PROBLEMA	Los desarrolladores deben ser capaces de establecer su factor de carga individual, porcentaje de tiempo que dedicaran realmente al desarrollo de las mismas.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos.

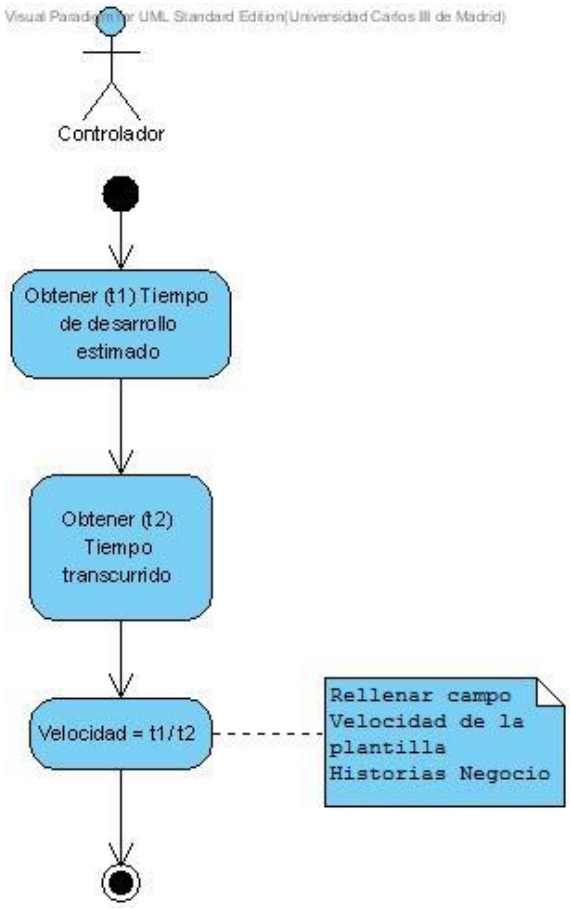
	Solución: 
ROLES	• Desarrolladores (2 a 12)
ENTRADAS	• Tareas Historias versión X Programador Y
LECCIONES APRENDIDAS	• Los programadores deben establecer un valor numérico, factor de carga, que representara el porcentaje de tiempo que dedicaran realmente al desarrollo de las tareas. El factor de carga representa la relación entre el numero ideal de días de programación y los que han transcurrido. Para el establecimiento del factor de carga deben tenerse en cuenta el tiempo que se dedica a ayudar a otros programadores, hablar con el cliente y asistir a reuniones, entre otras. Si el factor de carga supera los 8 puntos significa que al desarrollador no le quedara tiempo para ayudar a nadie. • En el documento de salida *Tareas Historias versión X*

	Programador Y se rellena la cabecera del documento con el Factor de Carga del programador en cuestión. • Los desarrolladores establecen su factor de carga identificando si es demasiado alto o bajo.
PLANTILLAS	• Tareas_Historias_Ver_X_DyD_Y.doc
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Tareas Historias versión X Programador Y
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. • Para cada miembro del equipo es importante que conozcan su factor de carga individual, y el tiempo que realmente va a dedicar al desarrollo del sistema. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.

HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.13 Patrón Establecer Velocidad

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Establecer Velocidad
PATRONES RELACIONADOS	• Patrón Clasificar por Valor • Patrón Clasificar por Riesgo • Patrón Establecer Ámbito Versión
CONTEXTO INICIAL	Este producto puede usarse en proyectos en los que el área de tecnología debe comunicar al área de negocio la rapidez con la que el equipo de desarrollo puede programar en el tiempo ideal de ingeniería por mes, con el objeto de que estos últimos puedan establecer el alcance de una determinada versión del aplicativo a desarrollar. La velocidad se establece como las historias de usuario terminadas en cada iteración por el equipo.
CONTEXTO RESULTATE	El área de negocio conoce el número de puntos de historia (un punto de historia es una semana dentro de la estimación de cada historia) que le van a ser entregadas al final de las 3 semanas máximas de las estimaciones de las historias.
PROBLEMA	La velocidad debe ser obtenida mediante un método empírico.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Debe obtenerse siempre que el área de negocio requiera de este dato para poder establecer un alcance de una versión del aplicativo a desarrollar.

SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos. **Solución:** 
ROLES	• Controlador (1)
ENTRADAS	• Historias de Negocio

LECCIONES APRENDIDAS	• La velocidad le sirve al área de tecnología para limitar las peticiones del área de negocio en cuanto al establecimiento del ámbito de una versión. Si la velocidad no está correctamente establecida el equipo de desarrollo puede llegar a no cumplir los plazos. Lo mejor para establecer la velocidad es asignar a cada programador un punto de historia, por lo que si son 6 programadores, la velocidad será 6 puntos de historia cada semana. Si la velocidad se multiplica por las 3 semanas máximas de estimación se obtiene el número de puntos de historia máximo para el establecimiento del ámbito de una versión. • El documento de entrada es un documento de uso interno para desarrollo. Debe contener una tabla con la clasificación de las historias de negocio en función del valor y el riesgo. En éste documento se registrará la velocidad. • El documento de salida ha de tener el campo velocidad relleno.
PLANTILLAS	• Historias_Negocio.doc
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Historias de Negocio
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el

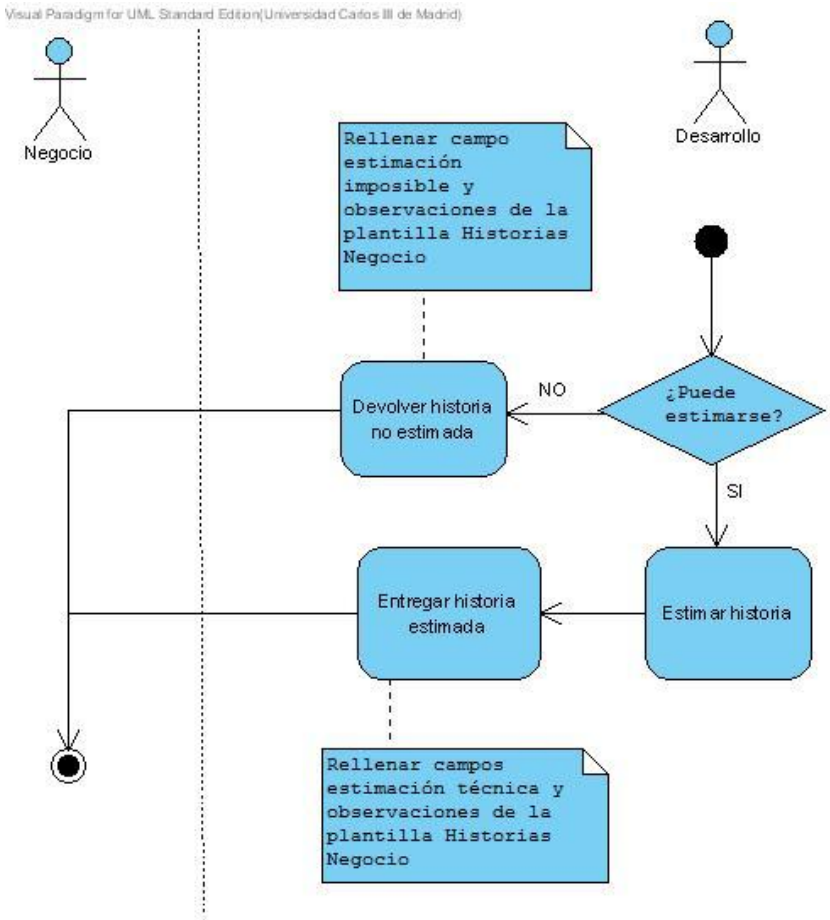
	programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La

	programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.14 Patrón Estimar Historia

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Estimar Historia
PATRONES RELACIONADOS	• Patrón Escribir Historia • Patrón Dividir Historia • Patrón Clasificar por Valor • Patrón Clasificar por Riesgo
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que una vez recogidos los requisitos del área de negocio para la implementación de un aplicativo, estos tengan que ser estimados por el área de tecnología. La estimación se refiere al tiempo, en semanas, que se estima va a tardar la implementación completa (análisis, diseño, desarrollo, pruebas, integración) de una funcionalidad (historia). La estimación más eficiente se basa en la estimación por puntos de historia, que es el único mecanismo que determina el tamaño de una historia de usuario, es decir, es la misma medida para cualquier miembro del grupo, con independencia de la experiencia en el equipo (desarrollador junior/desarrollador sénior). Los puntos de historia son el tamaño relativo de cada historia de usuario comparadas con otras historias estimadas por el mismo equipo.
CONTEXTO RESULTATE	Se obtendrán dos conjuntos de historias, unas que habrán sido estimadas por el área de tecnología y otras que no habrán podido ser estimadas debido a que son demasiado complejas para tratarlas unitariamente o no están suficientemente detalladas. En este último caso serán devueltas al área de negocio para que, o bien las simplifique (en varias historias) o las detalle completamente.

PROBLEMA	El área de tecnología debe conseguir estimar todas las historias entregadas por el área de negocio. La estimación máxima, para cada historia), no puede exceder las 3 semanas. Una vez que el equipo gane experiencia estimando en puntos de historia, la estimación se hace más precisa y puede ser realizada muy rápidamente en comparación con las técnicas tradicionales de estimación.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de Cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Se deben establecer reuniones entre las áreas tecnológicas y de negocio donde la comunicación sea fluida y haya confianza mutua. El espacio de trabajo debe estar organizado conforme a las directrices de la metodología. Es importante que no falte comida y bebida para que las reuniones sean distendidas.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 15 minutos.

	Solución: 
ROLES	• Desarrolladores (2 a 12) • Director de proyecto de desarrollo (1) • Preparador (1) • Controlador (1) Nota: el preparador y el controlador pueden ser la misma persona.
ENTRADAS	• Historias de Negocio
LECCIONES APRENDIDAS	• Debe realizarse la estimación en base a una implementación lo más sencilla posible. Se debe tener en cuenta que la propiedad del código generado es común a

<table>
<tr><td></td><td>todos los integrantes del proyecto. Si la historia no puede estimarse respondiendo a las anteriores premisas debe devolverse al área de negocio para que la reescriba.
• Establecer reuniones en las que siempre participe el área de negocio junto con el equipo de desarrollo (programadores incluidos).
• Estimaciones por puntos de historia más eficientes que los métodos tradicionales por su resultado homogéneo ante distintos miembros del equipo.</td></tr>
<tr><td>PLANTILLAS</td><td>• Historias_Negocio.doc</td></tr>
<tr><td>EJEMPLOS</td><td>Variables de historia:

A: Complejidad de historia (0.25)
B: Importancia en el negocio (0.75)

<table>
<tr><th>HISTORIA</th><th>A</th><th>B</th><th>TOTAL</th><th>PUNTOS DE HISTORIA</th></tr>
<tr><td>1. Hacer reserva teléfono</td><td>4</td><td>3</td><td>4*0.25+3*0.75=3.25</td><td>2</td></tr>
<tr><td>2.Hacer reserva TPR</td><td>6</td><td>10</td><td>6*0.25+10*0.75=9</td><td>5</td></tr>
<tr><td>3. CA1: Hacer reserva TPR</td><td>6</td><td>9</td><td>6*0.25+9*0.75=8.25</td><td>4</td></tr>
<tr><td>4. CA2: Hacer reserva TPR</td><td>6</td><td>9</td><td>6*0.25+9*0.75=8.25</td><td>4</td></tr>
<tr><td>5. Activar reserva TPR</td><td>4</td><td>10</td><td>4*0.25+10*0.75=8.5</td><td>5</td></tr>
<tr><td>6. CA1: Activar reserva TPR</td><td>4</td><td>8</td><td>4*0.25+8*0.75=7</td><td>4</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>7. Hacer reserva maître</td><td>6</td><td>10</td><td>6*0.25+10*0.75=9</td><td>5</td></tr>
<tr><td>8. CA1: Hacer reserva maître</td><td>9</td><td>7</td><td>9*0.25+7*0.75=7.5</td><td>4</td></tr>
</table>
</td></tr>
<tr><td>SALIDAS</td><td>• Historias de Negocio</td></tr>
<tr><td>CONTROLADORES DE CALIDAD</td><td>• Ninguno</td></tr>
<tr><td>NIVEL DE MADUREZ</td><td>Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).</td></tr>
<tr><td>CONOCIMIENTOS BÁSICOS</td><td>Conocimientos:

• Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona.
• Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades.
• Es necesario conocer la complejidad de cada historia de usuario así como la importancia en el negocio para poder realizar una buena estimación por puntos de historia.

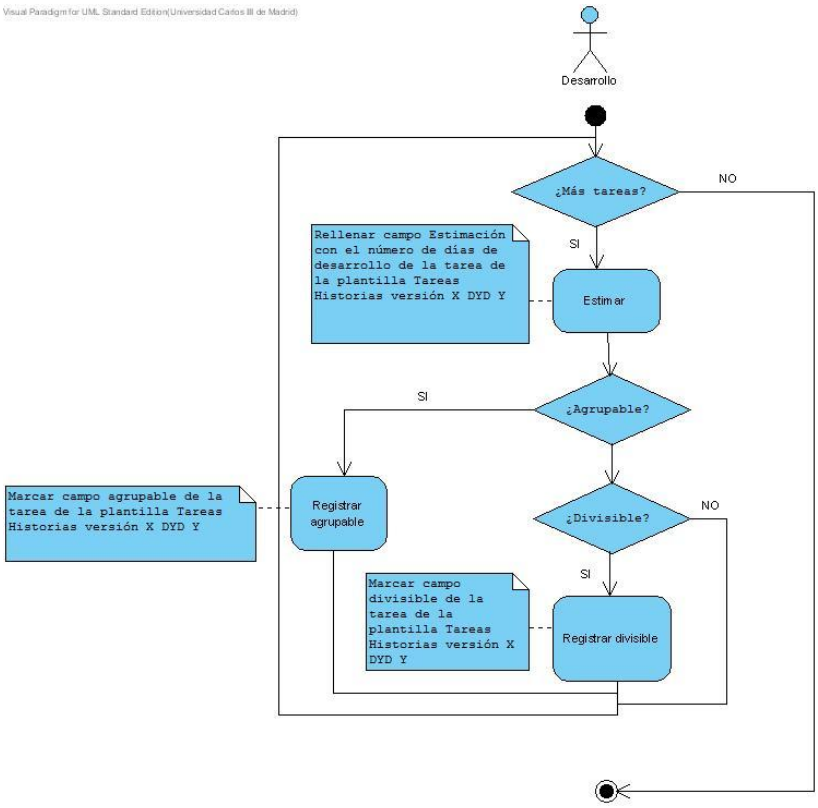
Habilidades:

• Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden.
• Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después.
• Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.</td></tr>
</table>

HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.15 Patrón Estimar Tarea

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Estimar Tarea
PATRONES RELACIONADOS	• Patrón Asignar Tarea • Patrón Dividir Tarea • Patrón Unir Tarea • Patrón Establecer Factor de Carga
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que deban estimarse las tareas técnicas a realizar resultantes de la transformación de las historias de negocio en dichas tareas.
CONTEXTO RESULTATE	Los desarrolladores obtienen una estimación del número ideal de días de ingeniería para implementar cada una de las tareas que les han sido encomendadas.
PROBLEMA	Los desarrolladores deben ser capaces de estimar cada tarea, y de dividir las tareas que necesiten más de unos días en ser implementadas o bien agruparlas.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Pueden usarse técnicas de estimación basadas en Puntos de función de Albretch y COCOMO.
	Tiempo de Desarrollo: • Para crear el Patrón de Producto: 45 minutos.

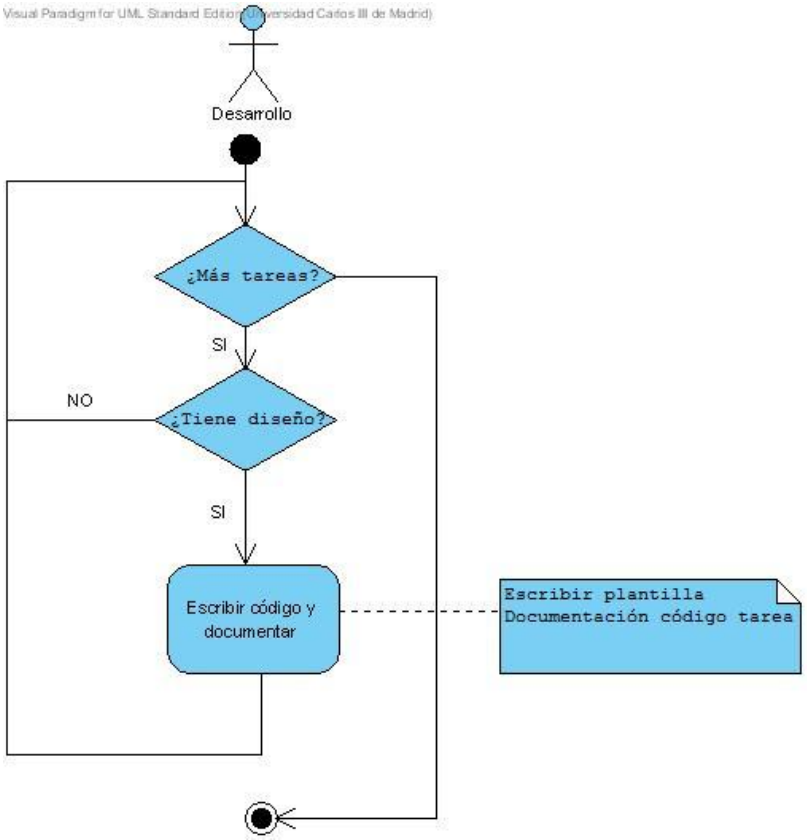
SOLUCIÓN	**Solución:** 
ROLES	• Desarrolladores (2 a 12)
ENTRADAS	• Tareas Historias versión X Programador Y
LECCIONES APRENDIDAS	• El programador debe estimar las tareas que le tocan, y en función del resultado de la estimación, comparando las tareas que acaban en tiempo con las que no, debe decidir si se dividen en otras más sencillas o si por el contrario se agrupan. • Los desarrolladores estiman las tareas y deciden su división o agrupación comparando las tareas que acaban en tiempo con las que no. • El documento de salida *Tareas Historias versión X Programador Y* es de uso exclusivo para el programador en cuestión con las tareas que le han sido asignadas. En caso de que la estimación de la tarea no sea asumible se marcara el campo correspondiente. En caso de que la

	tarea sea muy sencilla se marca el campo correspondiente.
PLANTILLAS	• Tareas_Historias_Ver_X_DyD_Y.doc
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Tareas Historias versión X Programador Y
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de

RECURSOS DE INFORMACIÓN	http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.16 Patrón Generar Código

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Generar Código
PATRONES RELACIONADOS	• Patrón Diseñar • Patrón Registrar Progreso
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que se siga la filosofía de trabajo en parejas para la implementación de código. En XP todo el software productivo se escribe en pareja, dos programadores sentados lado a lado en una misma computadora. Esta práctica asegura que todo el código productivo fue revisado por al menos otro programador, y genera mejores diseños, mejores pruebas y mejor código. Puede parecer ineficiente que dos programadores hagan el "trabajo de un programador", pero lo contrario es cierto. Los estudios sobre la programación de a pares muestran que las parejas producen mejor código en aproximadamente el mismo tiempo que un programador trabajando solo.
CONTEXTO RESULTATE	Se obtienen pequeñas piezas de código implementadas por dos personas, lo cual reduce el porcentaje de errores (principio de los cuatro ojos). Los equipos XP usan un estándar de código en común, de manera que el código del sistema se vea como si fuera escrito por una única persona muy competente. No importa mucho el estándar en si mismo: lo importante es que el código se vea familiar, para permitir la propiedad colectiva.
PROBLEMA	La programación en parejas no consiste en que una persona escriba y la otra mire, es un dialogo entre dos personas que intentan simultáneamente programar, diseñar, analizar y probar.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de

	usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos. **Solución:** 
ROLES	• Desarrolladores (2 a 12)

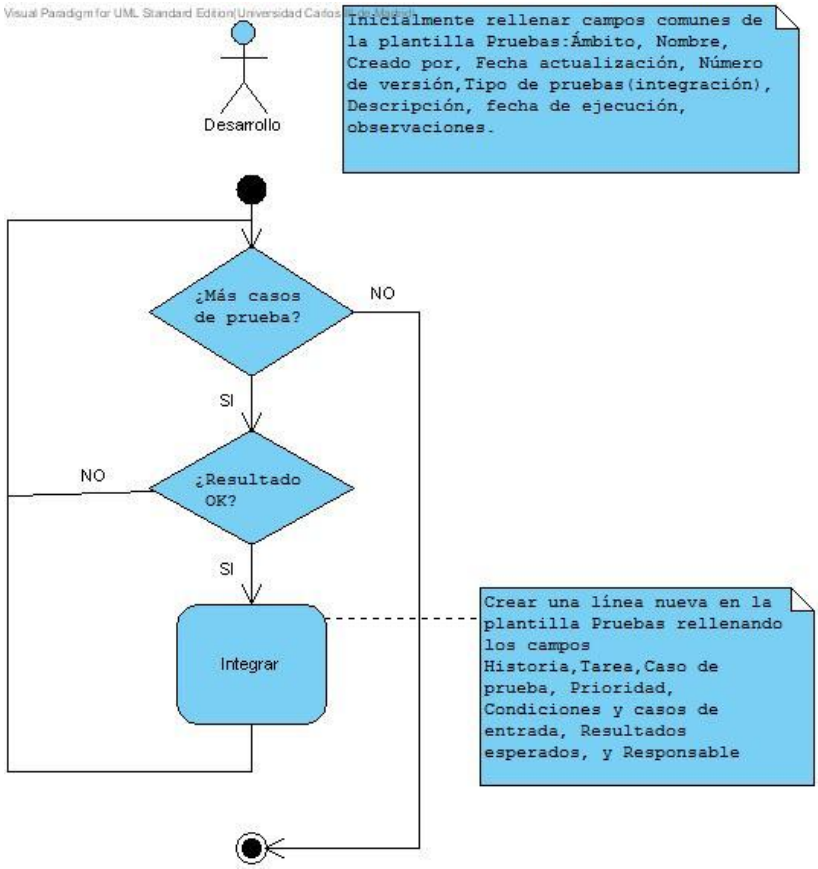
ENTRADAS	• Diseño Tarea
LECCIONES APRENDIDAS	• La programación en parejas no es una tutoría. Si en la pareja hay un programador experto y otro inexperto, el primero debe asesorar y guiar al otro para que al cabo de un corto tiempo alcance el nivel requerido. Las parejas de desarrolladores deben estar bien organizadas. Si se detecta que la relación entre los miembros de una pareja no es buena, conviene perder un cierto tiempo en la reorganización de las mismas. Lo mas importante para que el trabajo en equipo funcione es la comunicación entre los miembros del la pareja de programadores. El código implementado debe ser sencillo, para lo cual se deben seguir las recomendaciones: 1. El sistema (código y pruebas) debe comunicar todo lo que se quiere comunicar y nada mas. 2. El sistema no debe tener código duplicado 3. El sistema debería obtener el menor numero posible de clases 4. El sistema debería obtener el menor numero posible de métodos Debemos realimentarnos de lo aprendido en el diseño. Se debe hacer de forma cíclica todo el proceso tan pronto como sea posible: 5. Inversión inicial pequeña 6. Asumir la simplicidad 7. Cambio incremental 8. Viajar ligero (no intentar cubrir expectativas que no esta descritas en el diseño) • Conviene escribir código de manera conjunta entre dos personas que colaboran entre sí. • El archivo *Diseño Tarea* es un documento con el diseño de la tarea a implementar. • El producto de salida *Código fuente* es el código fuente que implementa la tarea en cuestión. • El documento *Documentación Código Fuente* es la documentación que explica el código implementado.

PLANTILLAS	• Codigo_Tarea.xxx • Documentación_Codigo_Tarea.doc
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Código Tarea • Documentación Código Tarea
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.

RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.17 Patrón Integrar

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Integrar
PATRONES RELACIONADOS	• Patrón Pruebas Unitarias • Patrón Modificar Casos de Prueba • Patrón Pruebas de Integración
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en los que tras el diseño u la implementación de código o modificación del mismo deba integrarse con el resto de trabajo realizado por el equipo.
CONTEXTO RESULTATE	El trabajo realizado por un miembro del equipo funciona 100% con el resto de trabajos realizado por todo el equipo.
PROBLEMA	Los desarrolladores deben mantener equilibrado e integrado su trabajo con el del resto del equipo. La integración continua implica que debe integrarse el trabajo individual cada pocas horas (1 día como máximo). En la integración se juntan los pequeños módulos de software se tienen y que parece que funcionan, pero resulta que esos pequeños trozos solo han superado pruebas de unidad donde no han tenido que interactuar con otros módulos. El problema es que existen errores que solo surgen en esa interacción y ese es precisamente el problema de la integración.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.

SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos. **Solución:** 
ROLES	• Desarrolladores (2 a 12)
ENTRADAS	• Pruebas
	• Es importante tener una herramienta de integración que soporte un ciclo rápido de integración/construcción/pruebas. El conjunto de pruebas deben ejecutarse en unos pocos minutos. Además, las colisiones deben requerir un esfuerzo relativamente

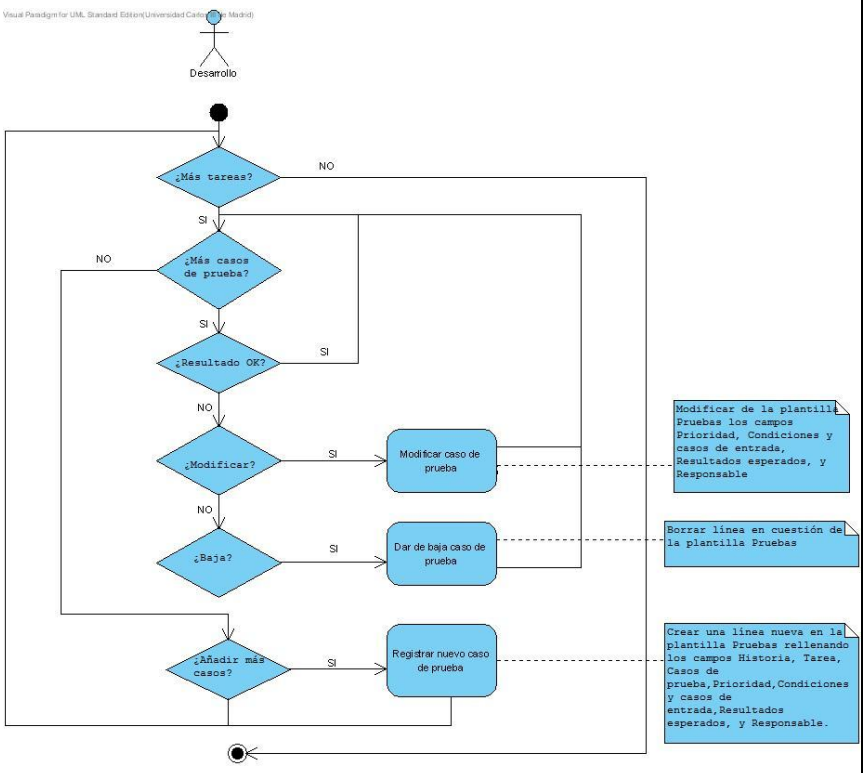
LECCIONES APRENDIDAS	pequeño. Se reduce de manera drástica el riesgo del proyecto. Si dos personas tienen diferentes ideas sobre la forma o el funcionamiento de una parte del código se sabrá en horas. Nunca se dedicarán muchas horas a perseguir un error que fue cometido en algún momento de las últimas semanas. • Hay que integrar cada vez que se desarrolle una clase o método para mantener el equilibrio y la unidad de todo el trabajo conjunto del equipo. • El archivo *Pruebas* es un documento genérico, para todos los tipos de prueba con la casilla pruebas unitarias marcada, que aglutina todos los casos de prueba para una tarea en cuestión. Hay que poner especial atención en aquellos casos de prueba no satisfactorios, pues han de ser modificados. En este documento se refleja su responsable e historia asociada. Se pueden añadir o eliminar casos de prueba según se considere oportuno.
PLANTILLAS	• Pruebas.xls
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Código Tarea • Documentación Código Tarea
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades.

	Habilidades: • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La

	programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.18 Patrón Modificar Casos de Prueba

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Modificar Casos de Prueba
PATRONES RELACIONADOS	• Patrón Pruebas Unitarias
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que una vez ejecutados los casos de prueba alguno de estos no es satisfactorio porque no se haya contemplado alguna casuística en concreto.
CONTEXTO RESULTATE	Los desarrolladores obtienen casos de prueba modificados, añadiendo o quitando elementos, ya sea por falta de amplitud en su alcance o por exceso.
PROBLEMA	Los desarrolladores deben ser capaces de averiguar si el caso de prueba identificado es completo y determinante para, en caso contrario, poder modificarlo adecuadamente. La modificación de los casos de prueba puede realizarse bajo estas circunstancias, entre otras: 1. Si se encuentra un problema, se escribe una prueba que aísla el mismo. 2. Si se está haciendo recodificación en algún código, y no se está seguro de cómo debe ser su comportamiento, y todavía no hay una prueba para dicho comportamiento en cuestión, escribe una primera prueba.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en

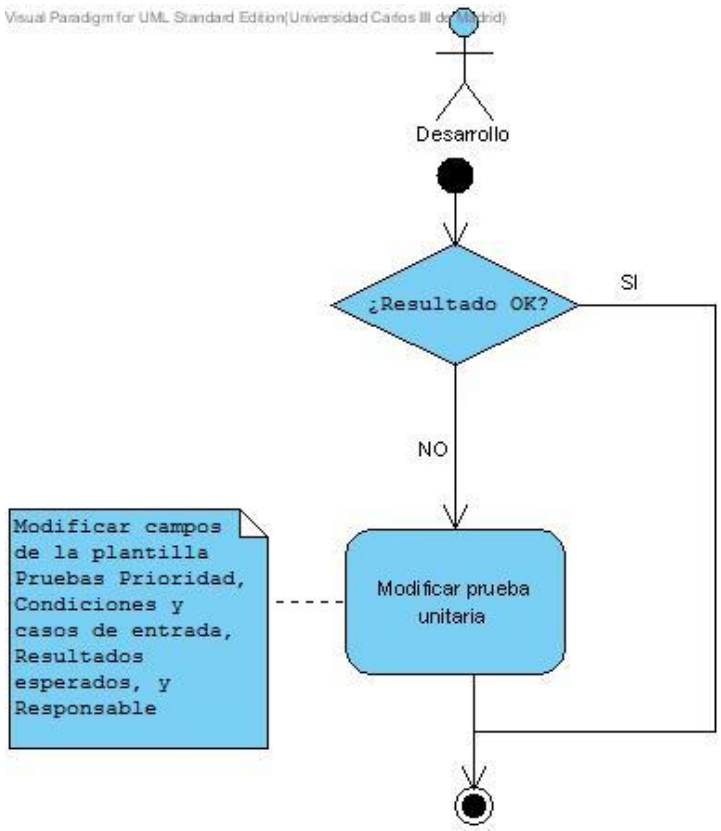
	la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos. **Solución:** 
ROLES	• Desarrolladores (2 a 12)
ENTRADAS	• Pruebas
LECCIONES APRENDIDAS	• La pareja de programadores debe volver a analizar la tarea relacionada con el caso de prueba no satisfactorio. La actividad de corregir los casos de prueba es la más importante de las que tienen que realizar los miembros del equipo de trabajo, aunque debe mantenerse el

	equilibrio entre el tiempo que se dedica a la ejecución de las pruebas y a la corrección de las mismas. • Si el caso de prueba no es satisfactorio debe volver a analizarse la tarea para modificar los casos de prueba identificados. • El archivo *Pruebas* es un documento genérico, para todos los tipos de prueba con la casilla pruebas unitarias marcada, que aglutina todos los casos de prueba para una tarea en cuestión. Hay que poner especial atención en aquellos casos de prueba no satisfactorios, pues han de ser modificados. Se pueden añadir o eliminar casos de prueba según se considere oportuno.
PLANTILLAS	• Pruebas.xls
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Código Tarea • Documentación Código Tarea
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de

	generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica.Ed Addisson Wesley.

12.4.19 Patrón Modificar Integración

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Modificar Integración
PATRONES RELACIONADOS	• Patrón Pruebas de Integración
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que este presente el concepto de integración continua y propiedad colectiva, por el cual cualquier miembro del equipo de trabajo puede modificar lo realizado por otro miembro.
CONTEXTO RESULTATE	Los desarrolladores obtienen conclusiones de las pruebas de integración que no fueron satisfactorias o que pueden ser simplificadas.
PROBLEMA	Los desarrolladores deben mantener los principios de la integración continua para que nunca se pierda demasiado tiempo en la identificación de un problema causado por un error cometido en las últimas semanas. En la integración deben tenerse en cuenta que el sistema completo (código y pruebas) debe poder ser revisado por cualquier miembro del equipo.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
	Tiempo de Desarrollo: • Para crear el Patrón de Producto: 45 minutos.

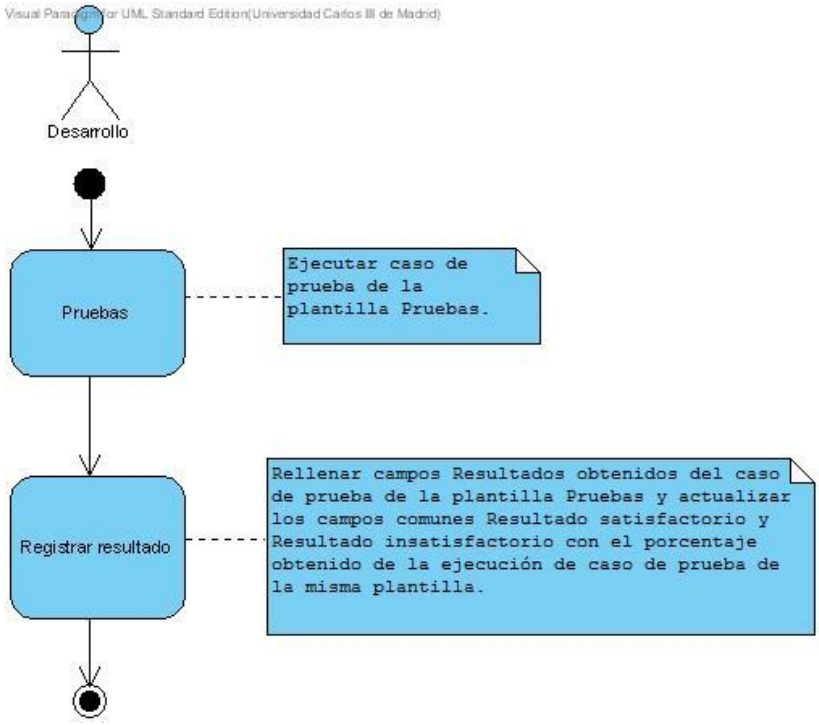
SOLUCIÓN	**Solución:** 
ROLES	• Desarrolladores (2 a 12)
ENTRADAS	• Pruebas
LECCIONES APRENDIDAS	• Recodificar continuamente tiene el efecto de fraccionar el sistema en montones de pequeños objetos y montones de pequeños métodos. Esto disminuye la probabilidad de que dos parejas de programadores cambien la misma clase o el mismo método al mismo tiempo. Si lo hacen, el esfuerzo requerido para reconciliar los cambios es pequeño, porque cada uno representa solo unas pocas horas de desarrollo. • La modificación de la integración en función de pruebas no satisfactorias o identificación de elementos que puede simplificarse más aun. • El archivo *Pruebas* es un documento genérico, para todos los tipos de prueba con la casilla pruebas unitarias

	marcada, que aglutina todos los casos de prueba para una tarea en cuestión. Hay que poner especial atención las pruebas de integración no satisfactorias, pues han de ser modificadas. En este documento se refleja su responsable e historia asociada. Se rellena el campo correspondiente al resultado de la prueba y las observaciones asociadas.
PLANTILLAS	• Integración_Continua.doc
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Pruebas
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for

	UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.20 Patrón Pruebas de Integración

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Pruebas Pruebas de Integración
PATRONES RELACIONADOS	• Patrón Integrar • Patrón Modificar Integración • Patrón Diseñar
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que una vez realizado el trabajo de integración del trabajo individual de un miembro del equipo de trabajo este deba probarse en conjunto con lo realizado por el resto de miembros del equipo. Las pruebas de integración se efectuarán siempre, antes de añadir cualquier nueva clase al proyecto, o después de modificar cualquiera existente.
CONTEXTO RESULTATE	Los desarrolladores obtienen un resultado que les muestra la capacidad que tiene su código de integrarse con el resto de trabajo de sus compañeros. Se verifica la duplicidad del código y redundancias con el trabajo de otros compañeros.
PROBLEMA	Las pruebas deben realizarse en pocos minutos. Unas pruebas de integración que sean largas restan tiempo al programador para seguir con el desarrollo de su trabajo y por tanto con la identificación, formulación y adición de nuevos casos de pruebas a la integración.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.

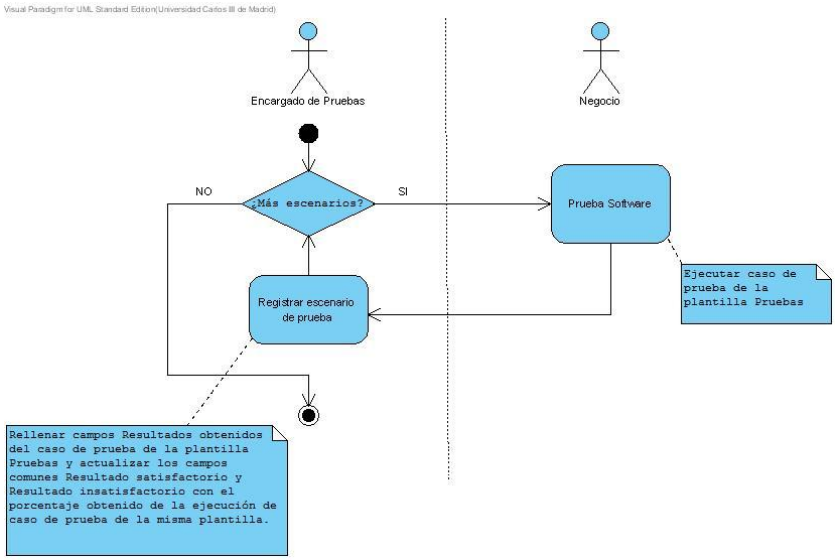
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos. **Solución:** 
ROLES	• Desarrolladores (2 a 12)
ENTRADAS	• Pruebas
LECCIONES APRENDIDAS	• Deben realizarse estas pruebas cada pocas horas (1 día como máximo). Cada vez que se está trabajando en una tarea, se tienen cientos de cosas en la mente. Se sigue trabajando hasta que hay una ruptura natural (no hay más elementos en la ficha de tareas a realizar) y entonces se hace la integración y sus pruebas asociadas. El ciclo debe ser Aprender/probar/programar/versión. • El archivo *Pruebas* es un documento genérico, para todos los tipos de prueba con la casilla pruebas unitarias marcada, que aglutina todos los casos de prueba para una tarea en cuestión. Hay que poner especial atención en aquellos casos de prueba no satisfactorios, pues han de

	ser modificados. En este documento se refleja su responsable e historia asociada. Se rellena el campo correspondiente al resultado de la prueba y las observaciones asociadas. • Hay que probar de manera conjunta el trabajo individual con lo realizado por el resto de miembros del equipo.
PLANTILLAS	• Integración_Continua.doc
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Pruebas
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for

	UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica.Ed Addisson Wesley.

12.4.21 Patrón Pruebas Funcionales Historia

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Pruebas Funcionales Historia
PATRONES RELACIONADOS	• Patrón Escribir Pruebas Funcionales para Historia • Patrón Escribir Casos de Prueba • Patrón Transformar Historia en Tareas • Patrón Escribir Historia
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que deban realizarse pruebas funcionales del software generado. A partir de las historias de usuario se hacen las pruebas funcionales y a cada una le puede corresponder una varias. No se considera que una historia ha sido completada hasta que no pase todas sus pruebas funcionales. Las pruebas funcionales deben hacerse antes de empezar a depurar.
CONTEXTO RESULTATE	El encargado de pruebas y los clientes obtienen una visión del porcentaje de pruebas satisfactorias para poder así establecer el grado de avance del proyecto.
PROBLEMA	Debe proporcionarse al cliente una herramienta lo suficientemente potente como para poder traducir sus ideas a un conjunto de escenarios sobre los que probar el software. Además, dicha herramienta debe permitir la ejecución y mantenimiento de los escenarios descritos por las ideas del cliente.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.

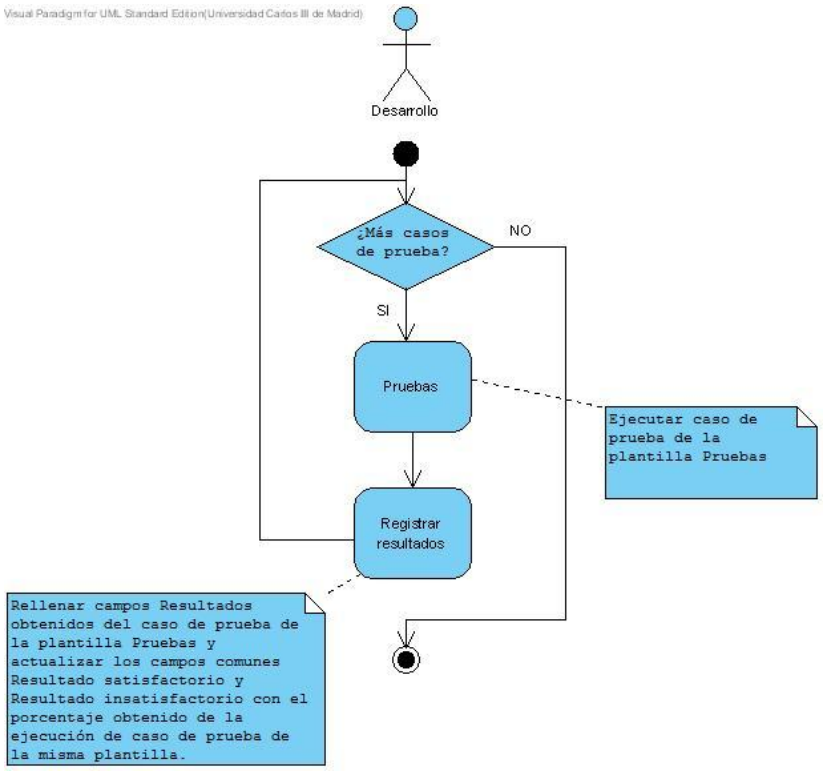
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos. **Solución:** 
ROLES	• Encargado de pruebas (1) • Usuarios del área de negocio (2 como mucho)
ENTRADAS	• Pruebas
LECCIONES APRENDIDAS	• No todas las pruebas funcionales tienen que funcionar al 100% al mismo tiempo. El resultado de las pruebas funcionales no puede ser binario, como en el caso de las pruebas unitarias, ya que provienen de distintos códigos generados por distintos desarrolladores del equipo de trabajo. Con el tiempo, rectificando las pruebas funcionales, el resultado de aproximarse al 100%. Conforme se vaya cerrando una versión, el cliente necesitara clasificar las pruebas funcionales que vayan fallando, estableciendo una prioridad para su corrección. • El cliente utiliza el software generado en cada uno de los escenarios descritos. • El documento de entrada *Pruebas* es un archivo .xls (Microsoft Excel) genérico, para todos los tipos de prueba con la casilla pruebas usuario marcada, que recoge la

	traducción de las ideas del cliente, en el lenguaje del ámbito técnico, a escenarios de prueba del software generado. En este documento, a la salida, se añade el porcentaje de satisfacción del cliente después de haber ejecutado las pruebas funcionales correspondientes. Además se incluye la prioridad para la corrección de las pruebas funcionales que no fueron satisfactorias.
PLANTILLAS	• Pruebas.xls
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Pruebas
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word.

HERRAMIENTAS DE SOPORTE	• Será necesario un editor de hojas de cálculo como OpenOffice Calc o Microsoft Excel. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.22 Patrón Pruebas Unitarias

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Pruebas Unitarias
PATRONES RELACIONADOS	• Patrón Escribir Casos de Prueba • Patrón Integrar • Patrón Modificar Casos de Prueba
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que se deban probar por separado las tareas a implementar de acuerdo a un conjunto de casos de prueba previamente identificados. Esto indica al programador que es lo que tiene que hacer cuando codifica. Los requisitos aparecen en forma de pruebas de unidad. Se sabe cuándo se ha terminado porque se superan todos los test de unidad. El beneficio que tiene de ello el diseño, es que en los test de unidad se pone aquello que es importante para el cliente.
CONTEXTO RESULTATE	Los desarrolladores obtienen una serie de resultados unitarios para cada caso de prueba asociado a una tarea.
PROBLEMA	Los desarrolladores deben ser capaces de conseguir que las pruebas unitarias de los casos de prueba, por separado, sean satisfactorias para poder continuar con el proceso. Un pequeño problema es que los test en si mismo pueden tener errores y esto conduce a que las pruebas unitarias no realicen su actividad de forma correcta.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.

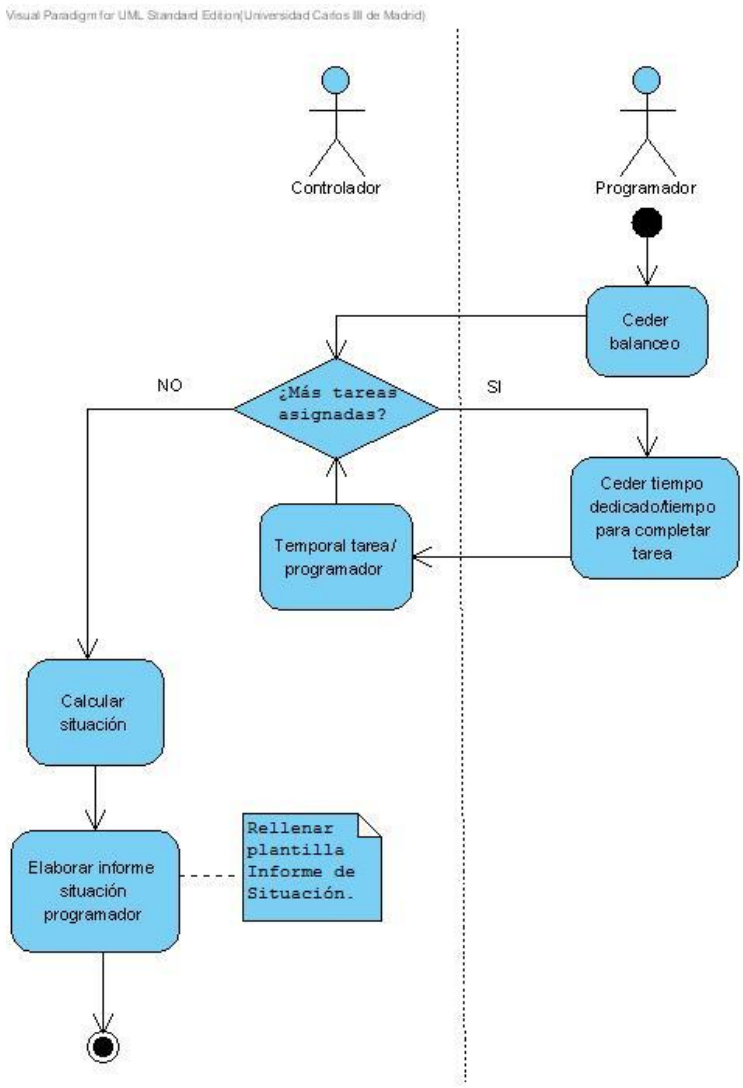
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos. **Solución:** 
ROLES	• Desarrolladores (2 a 12)
ENTRADAS	• Pruebas.xls
LECCIONES APRENDIDAS	• Si no funcionan las pruebas unitarias hay que corregirlas lo antes posible. Este es el trabajo más importante del equipo. Debido a esto, hay que mantener el equilibrio en el tiempo que se dedica a la realización de las pruebas y a la corrección de las que no sean satisfactorias. • La pareja de desarrolladores realizan pruebas unitarias sobre cada uno de los casos de prueba por separado. • El archivo *Pruebas.xls* es un documento genérico, para todos los tipos de prueba con la casilla pruebas unitarias

	marcada, que aglutina todos los casos de prueba para una tarea en cuestión.
PLANTILLAS	• Pruebas.xls
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Pruebas
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Será necesario un editor de hojas de cálculo como OpenOffice Calc o Microsoft Excel. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.

RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.23 Patrón Recuperación Tareas

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Recuperación Tareas
PATRONES RELACIONADOS	• Patrón Registrar Progreso • Patrón Asignar Tarea
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto por sobrecarga de trabajo de los programadores, deben tomarse decisiones que equilibren la carga de trabajo en el equipo.
CONTEXTO RESULTATE	Informe de situación con recomendaciones.
PROBLEMA	El controlador, en función del balanceo establecido por cada programador, y el resultado reflejado en el informe de situación con el registro del progreso, debe evaluar la situación y recomendar la toma de decisiones para asegurar la consecución del proyecto en los plazos fijados.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar**: Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos.

	Solución: 
ROLES	• Controlador (1)
ENTRADAS	• Tareas Historias versión X (Indicadores de Progreso / Informe de Seguimiento) • Tareas Historias versión X Programador Y Tipo: Documento Microsoft Word (.doc). Documento de uso exclusivo para el programador en cuestión con las tareas que le han sido asignadas. Se rellena la cabecera del documento con el balanceo establecido por el programador en cuestión

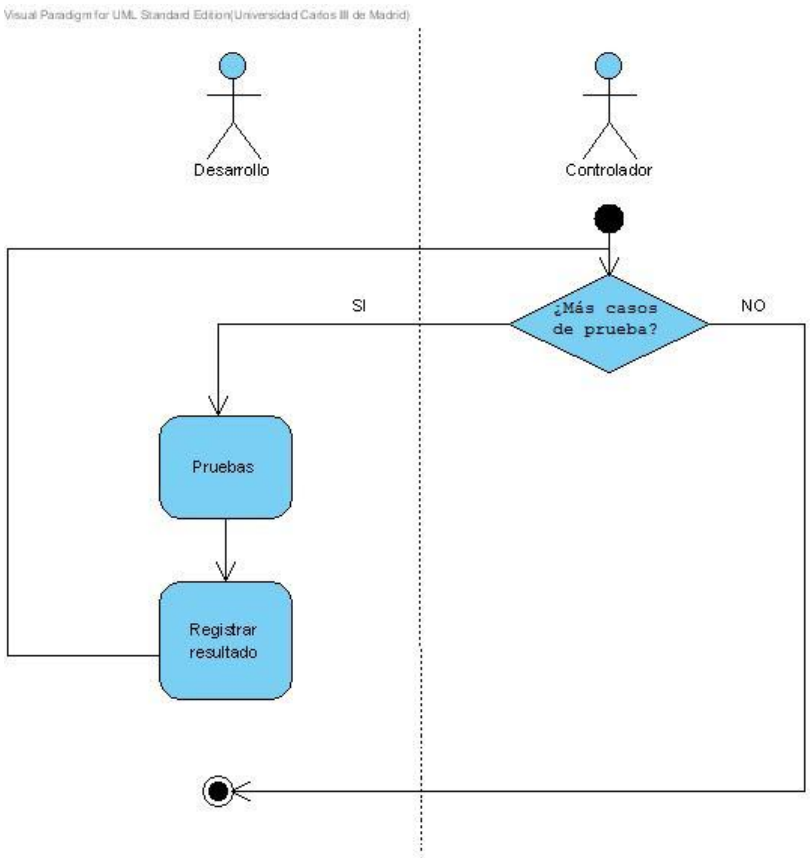
LECCIONES APRENDIDAS	• El controlador reflejara en el informe de situación la recomendación para el satisfactorio desarrollo del proyecto en caso de que el equipo se encuentre comprometido en exceso. Las recomendaciones pueden ser: 1. Reducir el ámbito de algunas tareas. 2. Pedir al cliente que reduzca el ámbito de algunas historias. 3. Quitar tareas no esenciales. 4. Programadores que obtengan mas o mejor ayuda y como último recurso. 5. Pedir al cliente que retrase algunas historias a una iteración posterior. • El documento de entrada *Tareas Historias versión X* contiene las tareas del ámbito técnico correspondientes a las historias de la versión en cuestión. Tanto el campo de *responsable de realización de cada tarea* como los de tiempo empleado en tarea y tiempo que resta para completarla has de estar rellenos. • El documento de entrada *Tareas Historias versión X Programador Y* es de uso exclusivo para el programador en cuestión y contiene las tareas que le han sido asignadas. Se rellena la cabecera del documento con el balanceo establecido por el programador en cuestión. • El documento de salida *Informe Situación* contiene recomendaciones para restablecer el equilibrio en el equipo de trabajo en caso de retrasos por sobrecarga de trabajo. • La elaboración del informe de situación se hace en base al balanceo de cada programador y al progreso de cada uno de ellos en la implementación de las tareas que les fueron encomendadas.
PLANTILLAS	• Informe_Situacion.doc
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Informe Situación
CONTROLADORES DE CALIDAD	• Ninguno

NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación

RECURSOS DE INFORMACIÓN	extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.24 Patrón Registrar Progreso

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Registrar Progreso
PATRONES RELACIONADOS	• Patrón Generar Código • Patrón Recuperación Tareas • Patrón Escribir Pruebas Funcionales para Historia
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que se deba registrar el progreso de los programadores en la implementación de las tareas que les fueron asignadas.
CONTEXTO RESULTATE	El controlador obtiene una visión clara del estado de desarrollo de la versión en cuestión.
PROBLEMA	El controlador debe establecer métricas adecuadas para el registro del progreso de cada programador.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos.

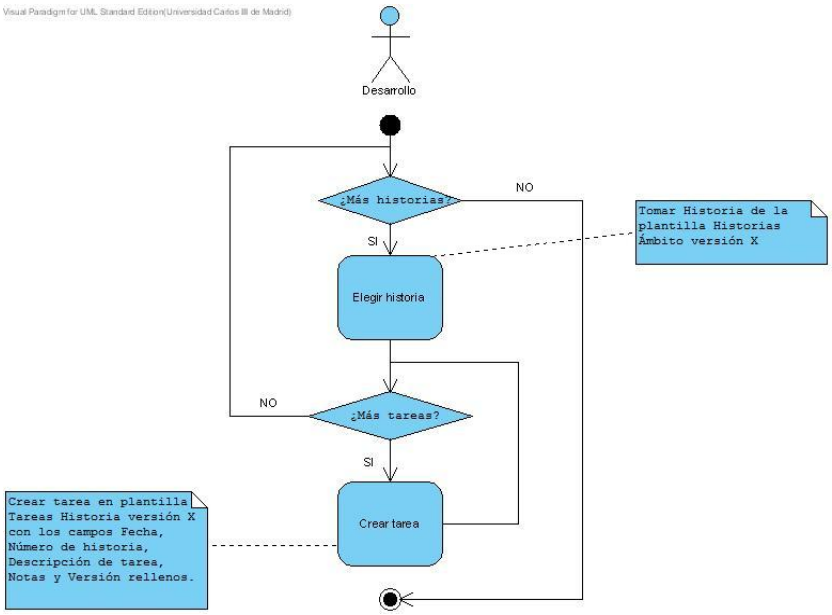
	Solución: 
ROLES	• Controlador (1)
ENTRADAS	• Tareas Historias versión X
LECCIONES APRENDIDAS	• El controlador debe establecer conclusiones a raíz de la recogida de información para poder ofrecer al director del proyecto un estado del desarrollo del mismo. Se deben realizar informes de seguimiento periódicamente para tener el control sobre el desarrollo del mismo. • El controlador pregunta a cada programador cuanto tiempo ha dedicado a cada una de sus tareas y cuanto le queda para completarlas. • El documento *Tareas Historias versión X* contiene las tareas del ámbito técnico correspondientes a las historias de la versión en cuestión. A la salida, el campo responsable de realización de cada tarea ha de estar relleno, así como los campos "tiempo empleado en tarea"

	y "tiempo que resta para completarla".
PLANTILLAS	• Tareas_Historias_Ver_X.doc
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Tareas Historias versión X (Indicadores de Progreso / Informe de Seguimiento)
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. • Se debe tener un conocimiento global del sistema y del trabajo del equipo para realizar un buen informe del estado de desarrollo. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.

RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.25 Patrón Transformar Historia En Tareas

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Transformar Historia En Tareas
PATRONES RELACIONADOS	• Patrón Establecer Ámbito Versión • Patrón Asignar Tarea
CONTEXTO INICIAL	Este producto puede usarse en todos los proyectos en los que deban trasladarse al ámbito técnico los requerimientos del área de negocio.
CONTEXTO RESULTATE	El área de tecnología obtiene un conjunto de tareas, en su lenguaje, sobre las que poder trabajar en la implementación de los requerimientos del área de negocio.
PROBLEMA	El área de tecnología debe ser capaz de, para las historias de negocio seleccionadas para la iteración en curso, transformarlas en tareas (1 o n por historia), del ámbito técnico. Las tareas deben ser lo suficientemente completas, y a la vez sencillas, para poder ser abordadas en unos pocos días.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos.

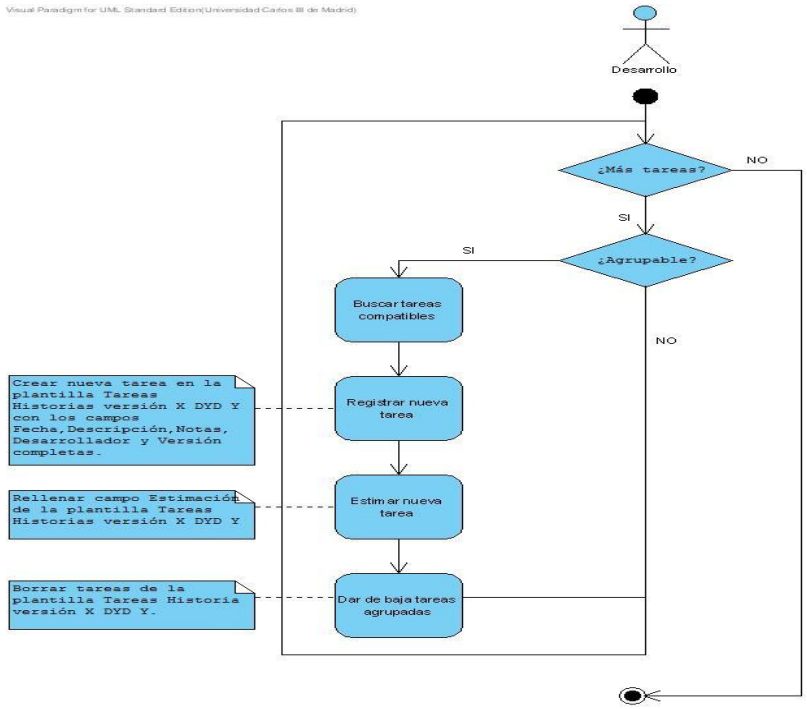
	Solución: 
ROLES	• Preparador (1) • Desarrolladores (2 a 12)
ENTRADAS	• Historias ámbito versión X
LECCIONES APRENDIDAS	• Deben transformarse las historias de negocio en tareas técnicas lo suficientemente completas y sencillas como para poder ser implementadas en unos pocos días. Si las tareas son demasiado complejas deben separase para ser implementadas en varias tareas. Si por el contrario son muy sencillas, deben agruparse para ser tratadas como una sola. Es posible que surjan tareas técnicas que no tengan nada que ver con las historias de negocio, pero que sin embargo son necesarias para poder implementar una tarea que si que tiene que ver con alguna historia. • Tecnología divide las historias de negocio en tareas del ámbito técnico para su implementación.
PLANTILLAS	• Tareas_Historias_Ver_X.doc
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.

SALIDAS	• Tareas Historias versión X
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.
RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com

	• Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.4.26 Patrón Unir Tarea

SECCIÓN	DESCRIPCIÓN
NOMBRE:	Unir Tarea
PATRONES RELACIONADOS	• Patrón Estimar Tarea
CONTEXTO INICIAL	Este producto puede usarse en cualquier proyecto en el que debido al resultado de la estimación de una tarea, por el cual se observe que puede completarse en unas pocas horas y que además pueda agruparse con otra/s tareas, se haga. Si la tarea en cuestión es demasiado sencilla o lleva muy poco tiempo realizarla será necesario combinarla con otra.
CONTEXTO RESULTATE	Los desarrolladores obtienen nuevas tareas como resultado de la agrupación de tareas sencillas.
PROBLEMA	Los desarrolladores deben ser capaces de estimar cada tarea, y de agruparlas entre si en el caso de que sea posible su implementación en unas pocas horas y sean complementarias.
RESTRICCIONES(FORCES)	• **Características de las organizaciones:** Este patrón puede utilizarse en los proyectos existentes en cualquier tipo de compañía. • **Tipo de sistema a desarrollar:** Este producto puede utilizarse en proyectos en los que los requerimientos de usuario sean cambiantes. • **Tipo de cliente:** Debe existir, o debe conseguirse, que el área de negocio destinataria del desarrollo se implique en la consecución del mismo. • **Heurísticas de uso:** Si se necesita disponer urgentemente del aplicativo o de algunas de sus funcionalidades.
SOLUCIÓN	**Tiempo de Desarrollo:** • Para crear el Patrón de Producto: 45 minutos.

	Solución: 
ROLES	• Desarrolladores (2 a 12)
ENTRADAS	• Tareas Historias versión X • Tareas Historias versión X Programador Y
LECCIONES APRENDIDAS	• En el documento de entrada *Tareas Historias versión X Programador Y* el campo indicando que la tarea es sencilla ha de estar marcado. • En el documento de salida *Tareas Historias versión X* se añaden al final de documento las nuevas tareas resultantes de la unión de las que tenían el campo en cuestión marcado, eliminando estas últimas tareas. • En el documento de entrada *Tareas Historias versión X Programador Y* se añaden al final de documento las nuevas tareas resultantes de la unión, eliminando estas últimas. • Los desarrolladores agrupan las tareas si la estimación y características de ellas lo permiten. • El programador debe agrupar un conjunto de tareas compatibles entre si y cuyas estimaciones no superen

	unas cuantas horas.
PLANTILLAS	• Tareas_Historias_Ver_X.doc • Tareas_Historias_Ver_X_DyD_Y.doc
EJEMPLOS	No hay ejemplos sobre el uso de este patrón de producto en este momento.
SALIDAS	• Tareas Historias versión X • Tareas Historias versión X Programador Y
CONTROLADORES DE CALIDAD	• Ninguno
NIVEL DE MADUREZ	Este Patrón de Producto no se relaciona con ningún nivel de madurez(N/A).
CONOCIMIENTOS BÁSICOS	**Conocimientos:** • Conocimiento del estándar de codificación que define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos. Los programadores las han de seguir de tal manera que el código en el sistema se vea como si hubiera estado escrito por una sola persona. • Conocimiento de la visión común de cómo funciona el programa en el que se desarrollan las actividades. **Habilidades:** • Capacidad de trabajo en grupo. Todos en un equipo XP contribuyen de la manera que pueden. • Predicción de qué se habrá terminado para la fecha de entrega, y determinación de qué hacer después. • Capacidad de programación de a pares. Además de generar mejor código y pruebas, sirve para comunicar el conocimiento a través de los equipos.
HERRAMIENTAS DE SOPORTE	• Será necesario un editor de texto como OpenOffice Writer o Microsoft Word. • Además de una herramienta como Visual Paradigm for UML para la realización de los diagramas expuestos.

RECURSOS DE INFORMACIÓN	• Álvarez, José R. y Arias Manuel. Método Extreme programming. Recuperado el 2010-03-05 de http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html • Anaya Villegas, Adrian. A propósito de programación extrema XP(extreme Programming).Recuperado el 2010-02-10 de http://www.monografias.com • Beck, K.(2000), *Una explicación de la programación extrema. Aceptar el cambio*. Ed. Addisson Wesley. • De Seta, Leonardo. Una introducción a Extreme Programming. Recuperado el 2010-03-02 de http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html • Extreme Programming: A gentle introduction. Recuperado el 2010-03-15 de http://www.extremeprogramming.org/ • Joskowicz, José. Reglas y prácticas en Xtreme Programming. Recuperado el 2010-03-15 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf • Letelier, Patricio y Panadés Mª Carmen. Metodologías Ágiles en el desarrollo de software: extreme programming. Recuperado el 2010-03-15 de http://www.willydev.net/descargas/masyxp.pdf • Newkirk, James y Martin, Robert C.(2001), La programación Extrema en la Práctica. Ed Addisson Wesley.

12.5 Fase tercera: Aplicación de los patrones de XP

La tercera fase del proyecto consiste en aplicar los patrones XP al caso práctico del restaurante. En este caso lo aplicaremos a la parte de pedidos.

Las actividades que realizaremos para esta parte serán, al igual que en la primera fase, la realización de las historias de usuario, la estimación de las mismas, así como los casos de prueba para cada una de ellas. Para ello los patrones consultados serán los que nos permitan realizar las actividades anteriormente descritas. Estos patrones son los siguientes:

- Patrón Asignar Tarea
- Patrón Clasificar por Riesgo
- Patrón Clasificar por Valor
- Patrón Dividir Historia
- Patrón Escribir Casos de Prueba
- Patrón Escribir Historia
- Patrón Estimar Historia
- Patrón Modificar Casos de Prueba

Para realizar las actividades pertenecientes a esta fase cogeremos los patrones destacados anteriormente e iremos siguiendo estos patrones para la realización de las mismas. En cada patrón podemos encontrar un diagrama de actividades. Siguiendo los pasos de cada uno de los diagramas seremos capaces de finalizar nuestra tarea siguiendo unos pasos establecidos.

Por ejemplo, para la realización de las historias de usuario contamos con un patrón llamado Escribir Historia, el cual cuenta con el siguiente diagrama de actividades:

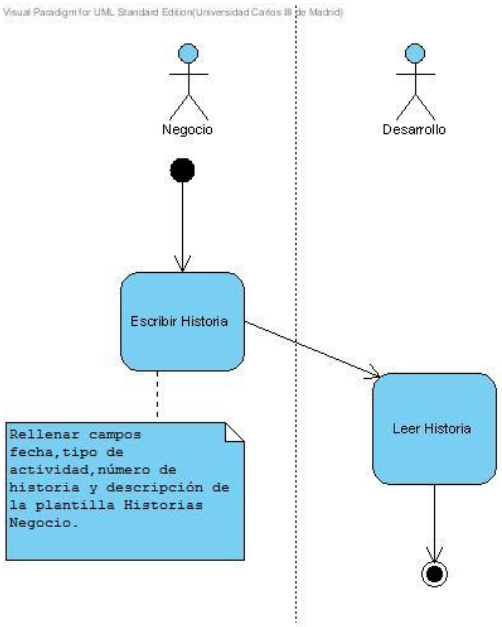


Figura 9: Diagrama de Actividad Escribir Historias

Si seguimos dicho diagrama la sucesión de actividades que deberemos realizar serán las siguientes:

Vemos que para la escritura de una historia los roles que se necesitan son dos: El personal de negocio y el desarrollador.

El personal de negocio escribe las historias, es decir, rellena los campos número de historia, nombre y descripción de la plantilla de historias de usuarios.

Esta historia pasa al desarrollador, el cual la lee y termina el proceso de escritura de historia. Después pasarán a escribir otra.

Una vez escritas las historias deberemos establecer la prioridad y el riesgo de la historia, para ello se utilizarán los patrones "Clasificar por Riesgo" y "Clasificar por Valor".

Al igual que hemos hecho para escribir las historias, iremos siguiendo los patrones para realizar esta tarea.

Una vez tenemos el riesgo y la priorización hecha tendremos que saber quien realiza la acción, por lo que deberemos asignar cada tarea a una o varias personas. Para ello utilizaremos el patrón "Asignar Tarea".

Además tenemos que asegurarnos de que las historias sean lo más sencillas posibles. Si no fuera el caso, necesitaremos el patrón "Dividir Historia" para hacer de cada una de ellas una parte sencilla de diseñar.

Después de esta breve explicación de cómo se han utilizado los patrones, se expondrán a continuación las historias de usuario de la parte de pedidos encontradas en el caso práctico con el que estamos trabajando, la gestión de un restaurante.

12.5.1 Realización de Historias De Usuarios

1. CREAR PEDIDO:

<table>
<tr><th colspan="3">Historia de Usuario</th></tr>
<tr><td>Número: 1</td><td colspan="2">Nombre: Crear Pedido</td></tr>
<tr><td colspan="3">Usuario: Camarero</td></tr>
<tr><td colspan="2">Modificación de Historia Número:</td><td>Iteración Asignada:</td></tr>
<tr><td colspan="2">Prioridad en Negocio: Alto
(Alta / Media / Baja)</td><td>Puntos Estimados:</td></tr>
<tr><td colspan="2">Riesgo en Desarrollo: Medio
(Alto / Medio / Bajo)</td><td>Puntos Reales:</td></tr>
<tr><td colspan="3">Descripción:
El camarero introduce el número de mesa y el código de las consumiciones de esta misma. Para toda consumición se comprueba que hay disponibilidad de todos los ingredientes para poder ser servida.
Cada consumición es anotada en una línea de pedido y no se podrá pasar a la siguiente consumición si ésta no ha sido validada antes por el sistema.</td></tr>
<tr><td colspan="3">Observaciones:</td></tr>
</table>

2. CASO ALTERNATIVO 1: CREAR PEDIDO

Historia de Usuario	
Número: **2**	Nombre: **Caso alternativo 1:Crear Pedido**
Usuario: Camarero	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Si no se pudiera completar una consumición por falta de ingredientes se le informará al cliente además de al almacén para que se solicite la reposición del mismo.	
Observaciones:	

3. CONSULTAR INGREDIENTES

Historia de Usuario	
Número: **3**	Nombre: **Consultar ingredients**
Usuario: Camarero	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alta (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Para cada consumición que el camarero vaya a anotar, se deberá hacer la comprobación de que existen los ingredientes necesarios para completar ésta misma. En el caso de que no exista se informará al almacén de que se necesita reponer y al cliente de que no puede disfrutar de esa consumición.	
Observaciones:	

4. CONGELAR PEDIDO

Historia de Usuario	
Número: **4**	Nombre: **Congelar pedido**
Usuario: Camarero	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Bajo (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: Bajo (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Una vez que los comensales han acabado de pedir sus consumidores, el camarero cierra temporalmente la nota pulsando un botón y ésta pasa a cocina que serán los encargados de preparar la comida pedida por los comensales.	
Observaciones:	

5 . REPONER INGREDIENTES

Historia de Usuario	
Número: **5**	Nombre: **Reponer ingredients**
Usuario: Encargado de cocina	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alta (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Cuando la comida pedida ya se ha preparado, desde la cocina se indica al sistema la disminución de los ingredientes que se han utilizado para la misma. Cuando el número de ingredientes ha disminuido tanto que está por debajo del límite el sistema deberá avisar directamente al almacén que los repongan.	
Observaciones:	

6. SERVIR PEDIDO

Historia de Usuario	
Número: **6**	Nombre: **Servir pedido**
Usuario: Camarero	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alta (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Cuando la comida esta lista para servir, el camarero será avisado mediante el sistema, desde cocina, para que la recoja y la sirva a los comensales, indicando que la mesa está servida.	
Observaciones:	

7. COBRAR NOTA

Historia de Usuario	
Número: **7**	Nombre: **Cobrar Nota**
Usuario: Camarero	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alta (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Cuando los comensales han acabado pedirán al camarero la nota, momento en el cual se cierra definitivamente el pedido. El camarero pedirá el impreso de la nota, la cual será entregada al cliente para su posterior pago, en efectivo o por tarjeta de crédito. Una vez pagada la cuenta, el camarero indicará el pago y establecerá la mesa como libre para posibles nuevas reservas.	
Observaciones:	

12.5.2 ESTIMACIÓN DE HISTORIAS DE USUARIOS

Una vez se han terminado de escribir las historias, aun queda un paso más, la estimación de las mismas.

Al igual que en el caso de la fase 1, cuando estimamos las historias de la parte de reservas del restaurante, vamos a utilizar la técnica de estimación por puntos de historia pero, en este caso además seguiremos las pautas que nos dan el patrón de "Estimar Historia".

Expondremos su diagrama de actividades para que se entienda mejor la explicación:

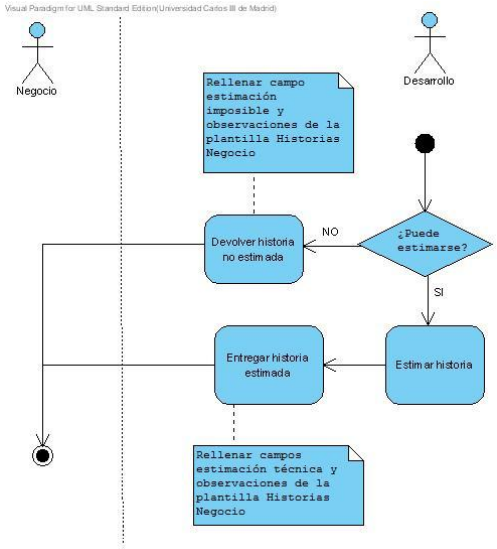


Figura 10: Diagrama de Actividad Estimar Historia

Como podemos observar en el diagrama, se necesitan dos roles para realizar esta actividad: El personal de negocio y el desarrollador.

El desarrollador, para cada historia mira si se puede estimar o no. Si no se puede estimar se termina la acción, pero si sí se puede, se estima y se entrega al personal de negocio.

Como en la fase primera, la estimación se hará en base a puntos de historia. La decisión de cómo se toman los valores de los puntos, es decir, como se determina porque una historia debe tener más o menos puntos de historia, es determinación de cada cliente.
Al no tener pautas de cuál de las historias debería tener una estimación de puntos de historia más alto que otra, hemos decidido, el ver un ejemplo en el patrón, establecer una priorización en la que el valor más alto será la historia que mayores puntos de historia tendrá.

La priorización, y por lo tanto la estimación de las historias de usuario de la parte de pedidos de nuestro restaurante será la siguiente:

Variables de historia:

A: Complejidad de historia (0.25)
B: Importancia en el negocio (0.75)

HISTORIA	A	B	TOTAL	PUNTOS DE HISTORIA
1. Crear Pedido	6	9	6*0.25+9*0.75=8.25	4
2. CA1: Crear Pedido	7	9	7*0.25+9*0.75=8.5	4
3. Consultar Ingredientes	6	8	6*0.25+8*0.75=6.5	3
4. Congelar Pedido	3	4	3*0.25+4*0.75=3.75	2
5. Reponer Ingredientes	7	10	7*0.25+10*0.75=9.25	5
6. Servir Pedido	6	9	6*0.25+9*0.75=8.25	4
7. Cobrar Nota	6	10	6*0.25+10*0.75=9	5

Una vez que hemos hallado los puntos de historia para cada una de ellas mostraremos a continuación las historias de usuario añadiendo este dato en la casilla de puntos estimados.

Por tanto las historias de usuario quedarán del siguiente modo:

1. CREAR PEDIDO:

Historia de Usuario	
Número: **1**	Nombre: **Crear Pedido**
Usuario: Camarero	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados: 4
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: El camarero introduce el número de mesa y el código de las consumiciones de esta misma. Para toda consumición se comprueba que hay disponibilidad de todos los ingredientes para poder ser servida. Cada consumición es anotada en una línea de pedido y no se podrá pasar a la siguiente consumición si ésta no ha sido validada antes por el sistema.	
Observaciones:	

2. CASO ALTERNATIVO 1: CREAR PEDIDO

Historia de Usuario	
Número: **2**	Nombre: **Caso alternativo 1:Crear Pedido**
Usuario: Camarero	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alto (Alta / Media / Baja)	Puntos Estimados: 4
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Si no se pudiera completar una consumición por falta de ingredientes se le informará al	

cliente además de al almacén para que se solicite la reposición del mismo.
Observaciones:

3.CONSULTAR INGREDIENTES

Historia de Usuario	
Número: **3**	Nombre: **Consultar ingredients**
Usuario: Camarero	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alta (Alta / Media / Baja)	Puntos Estimados: 3
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Para cada consumición que el camarero vaya a anotar, se deberá hacer la comprobación de que existen los ingredientes necesarios para completar ésta misma. En el caso de que no exista se informará al almacén de que se necesita reponer y al cliente de que no puede disfrutar de esa consumición.	
Observaciones:	

4.CONGELAR PEDIDO

Historia de Usuario	
Número: **4**	Nombre: **Congelar pedido**
Usuario: Camarero	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Bajo (Alta / Media / Baja)	Puntos Estimados: 2
Riesgo en Desarrollo: Bajo (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Una vez que los comensales han acabado de pedir sus consumidores, el camarero cierra	

temporalmente la nota pulsando un botón y ésta pasa a cocina que serán los encargados de preparar la comida pedida por los comensales.
Observaciones:

5 . REPONER INGREDIENTES

Historia de Usuario	
Número: **5**	Nombre: **Reponer ingredients**
Usuario: Encargado de cocina	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alta (Alta / Media / Baja)	Puntos Estimados: 5
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Cuando la comida pedida ya se ha preparado, desde la cocina se indica al sistema la disminución de los ingredientes que se han utilizado para la misma. Cuando el número de ingredientes ha disminuido tanto que está por debajo del límite el sistema deberá avisar directamente al almacén que los repongan.	
Observaciones:	

6.SERVIR PEDIDO

Historia de Usuario	
Número: **6**	Nombre: **Servir pedido**
Usuario: Camarero	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alta (Alta / Media / Baja)	Puntos Estimados: 4
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Cuando la comida esta lista para servir, el camarero será avisado mediante el sistema,	

desde cocina, para que la recoja y la sirva a los comensales, indicando que la mesa está servida.
Observaciones:

7.COBRAR NOTA

Historia de Usuario	
Número: **7**	Nombre: **Cobrar Nota**
Usuario:	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: Alta (Alta / Media / Baja)	Puntos Estimados: 5
Riesgo en Desarrollo: Medio (Alto / Medio / Bajo)	Puntos Reales:
Descripción: Cuando los comensales han acabado pedirán al camarero la nota, momento en el cual se cierra definitivamente el pedido. El camarero pedirá el impreso de la nota, la cual será entregada al cliente para su posterior pago, en efectivo o por tarjeta de crédito. Una vez pagada la cuenta, el camarero indicará el pago y establecerá la mesa como libre para posibles nuevas reservas.	
Observaciones:	

12.5.3 CASOS DE PRUEBA

Una vez hemos terminado de escribir las historias de usuario, además de estimarlas, pasamos a la elaboración de los casos de prueba para cada una de estas historias.

Como ya sabemos estos casos de prueba sirven para identificar errores antes de la implementación de dicha historia, evitando así contratiempos innecesarios.

Estos casos de prueba se realizan durante todo el proceso ya que un pequeño fallo en el diseño puede acarrear grandes desperfectos en la implementación.

Si se encontrase cualquier tipo de error al ejecutar estos casos de prueba, será resuelto de inmediato, evitando así problemas en fases más tardías.

Para la realización de estos casos de prueba se ha seguido como guía el patrón "Escribir Casos de Prueba".

Al igual que hemos hecho con los anteriores patrones utilizados, hemos seguido el diagrama de actividad descrito en el patón.

El diagrama de actividades de "Escribir Casos de prueba" es el siguiente:

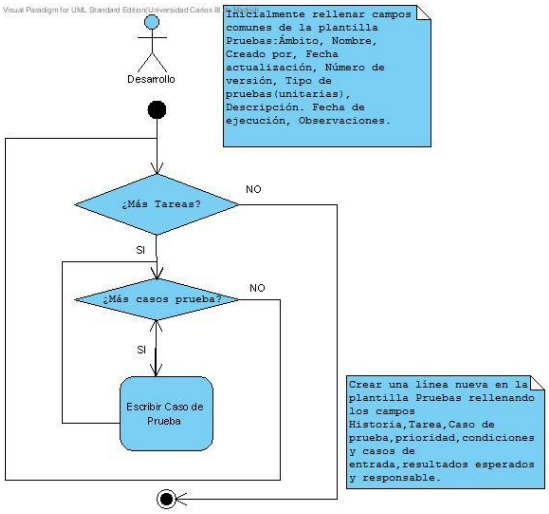


Figura 11: Diagrama de Actividad Escribir Casos de Prueba

Por tanto, siguiendo el diagrama mencionado, estos serán los casos de prueba para las historias de usuario de la parte de pedidos de la gestión del restaurante, es decir, de nuestro caso práctico.

1. Caso de Prueba 1: Crear Pedido

Caso de Prueba de Aceptación	
Código: 1	**Historia de Usuario (Nro. y Nombre): Crear Pedido**
Nombre: Crear Pedido	
Descripción: El camarero introduce el número de mesa y el código de las consumiciones de esta misma. Para toda consumición se comprueba que hay disponibilidad de todos los ingredientes para poder ser servida. Cada consumición es anotada en una línea de pedido y no se podrá pasar a la siguiente consumición si ésta no ha sido validada antes por el sistema.	
Condiciones de Ejecución: 1. Los clientes deben estar en la mesa reservada. 2. Los camareros le entregan la carta a los comensales.	
Entrada / Pasos de ejecución: 1. El camarero introduce el número de mesa en el terminal de pedidos. 2. Se graba automáticamente la hora en la que se hace el pedido. 3. El camarero introduce el código de la consumición que los comensales le piden y pulsa aceptar. 4. El sistema comprueba que hay ingredientes necesarios para satisfacer la petición. 5. Se repiten los dos pasos anteriores hasta que los comensales dejan de pedir. 6. El camarero cambia el estado de la mesa a “Esperar comida”. 7. El sistema avisa en cocina que hay un nuevo pedido a preparar.	
Resultado Esperado: Pedido creado y listo para preparar de una mesa.	
Evaluación de la Prueba:	

2. Caso de Prueba 2: Caso alternativo 1: Crear Pedido

Caso de Prueba de Aceptación	
Código: 2	**Historia de Usuario (Nro. y Nombre): CA1: Crear Pedido**
Nombre: CA1: Crear Pedido	
Descripción: Si no se pudiera completar una consumición por falta de ingredientes se le informará al cliente además de al almacén para que se solicite la reposición del mismo.	
Condiciones de Ejecución: 1. Los clientes deben estar en la mesa reservada. 2. Los camareros le entregan la carta a los comensales. 3. No se puede completar una consumición por falta de uno o varios ingredientes.	
Entrada / Pasos de ejecución: 1. El camarero introduce el número de mesa en el terminal de pedidos. 2. Se graba automáticamente la hora en la que se hace el pedido. 3. El camarero introduce el código de la consumición que los comensales le piden y pulsa aceptar. 4. El sistema comprueba que hay ingredientes necesarios para satisfacer la petición. 5. No se puede satisfacer dicha consumición por falta de uno o varios ingredientes, por lo que se le informa al almacén que se repongan. 6. El camarero informa al cliente de que no puede disponer de dicha consumición. 7. El camarero sigue introduciendo códigos de consumiciones y comprobando la disponibilidad en almacén de los ingredientes necesarios hasta que los comensales terminen de hacer el pedido. Si no hubiera algún ingrediente necesario se repetirían los pasos 5 y 6. 8. El camarero cambia el estado de la mesa a "Esperar Comida". 9. El sistema avisa en cocina que hay un nuevo pedido para preparar.	
Resultado Esperado: Pedido creado y listo para preparar de una mesa.	
Evaluación de la Prueba:	

3. Caso de Prueba 3: Consultar Ingredientes

Caso de Prueba de Aceptación	
Código: 3	**Historia de Usuario (Nro. y Nombre): Consultar Ingredientes**
Nombre: Consultar Ingredientes	
Descripción: Para cada consumición que el camarero vaya a anotar, se deberá hacer la comprobación de que existen los ingredientes necesarios para completar ésta misma. En el caso de que no exista se informará al almacén de que se necesita reponer y al cliente de que no puede disfrutar de esa consumición.	
Condiciones de Ejecución: 1. El camarero introduce al sistema el código de una consumición pedida por los comensales de una mesa. 2. El camarero pulsa aceptar.	
Entrada / Pasos de ejecución: 1. Los comensales piden al camarero una consumición. 2. El camarero introduce al sistema el código de la consumición pedida por los comensales y pulsa aceptar. 3. El sistema comprueba si hay ingredientes en el almacén para satisfacer dicha consumición.	
Resultado Esperado: Anotación de una consumición con ingredientes suficientes para satisfacerla.	
Evaluación de la Prueba:	

4. Caso de Prueba 4: Congelar Pedido

Caso de Prueba de Aceptación	
Código: 4	**Historia de Usuario (Nro. y Nombre): Congelar pedido**
Nombre: Congelar Pedido	
Descripción: Una vez que los comensales han acabado de pedir sus consumidores, el camarero cierra temporalmente la nota pulsando un botón y ésta pasa a cocina que serán los encargados de preparar la comida pedida por los comensales.	
Condiciones de Ejecución: 1. Los comensales terminan de pedir sus consumiciones.	
Entrada / Pasos de ejecución: 1. El camarero cierra temporalmente la nota, es decir, pulsa FIN. 2. El camarero cambia el estado de la mesa a "Esperar Comida". 3. El sistema avisa automáticamente a cocina de que hay un nuevo pedido para preparar.	
Resultado Esperado: Nota cerrada temporalmente.	
Evaluación de la Prueba:	

5. Caso de Prueba 5: Reponer Ingredientes

Caso de Prueba de Aceptación	
Código: 5	**Historia de Usuario (Nro. y Nombre): Reponer Ingredientes**
Nombre: Reponer Ingredientes	
Descripción: Cuando la comida pedida ya se ha preparado, desde la cocina se indica al sistema la disminución de los ingredientes que se han utilizado para la misma. Cuando el número de ingredientes ha disminuido tanto que está por debajo del límite el sistema deberá avisar directamente al almacén que los repongan.	
Condiciones de Ejecución: 1. Cocina debe haber preparado las bandejas que contienen el pedido de una mesa.	
Entrada / Pasos de ejecución: 1. Se indica al sistema que los ingredientes que contienen los platos han disminuido en cantidad. 2. El sistema comprueba si la cantidad de dichos ingredientes están por debajo del mínimo indispensable. 3. Si están por debajo de ese mínimo, el sistema avisa de que se repongan en almacén.	
Resultado Esperado: Ingredientes comprobados para cada consumición cocinada.	
Evaluación de la Prueba:	

6. Caso de prueba 6: Servir Pedido

Caso de Prueba de Aceptación	
Código: 6	**Historia de Usuario (Nro. y Nombre): Servir Pedido**
Nombre: Servir Pedido	
Descripción: Cuando la comida esta lista para servir, el camarero será avisado mediante el sistema, desde cocina, para que la recoja y la sirva a los comensales, indicando que la mesa está servida.	
Condiciones de Ejecución: 1. Los platos deben estar cocinados y listos para servir en la mesa. 2. Se ha disminuido el número de ingredientes, en almacén, necesarios para preparar dichos platos.	
Entrada / Pasos de ejecución: 1. El encargado de cocina establece el pedido como "Cocinado". 2. El encargado de cocina manda un mensaje al control del camarero para avisarle de que recoja el pedido. 3. El camarero recoge el pedido y lo lleva a la mesa correspondiente. 4. El camarero indica que la mesa está servida.	
Resultado Esperado: Pedido servido en mesa.	
Evaluación de la Prueba:	

7. Caso de prueba 7: Cobrar Nota

Caso de Prueba de Aceptación	
Código: 7	**Historia de Usuario (Nro. y Nombre): Cobrar Nota**
Nombre: Cobrar Nota	
Descripción: Cuando los comensales han acabado pedirán al camarero la nota, momento en el cual se cierra definitivamente el pedido. El camarero pedirá el impreso de la nota, la cual será entregada al cliente para su posterior pago, en efectivo o por tarjeta de crédito. Una vez pagada la cuenta, el camarero indicará el pago y establecerá la mesa como libre para posibles nuevas reservas.	
Condiciones de Ejecución: 1. Los comensales han terminado de consumir el pedido. 2. Los comensales piden la nota al camarero.	
Entrada / Pasos de ejecución: 1. El camarero cierra definitivamente el pedido de la mesa. 2. El camarero establece el estado de la mesa como “Esperando nota”. 3. El camarero ordena que se imprima la nota. 4. El camarero entrega la nota a los clientes. 5. Los clientes depositan el pago de la nota en efectivo o con tarjeta. 6. El camarero indica que la mesa está pagando. 7. El camarero entrega a los clientes la nota cobrada. 8. El camarero establece la mesa como “Libre”.	
Resultado Esperado: Nota cobrada.	
Evaluación de la Prueba:	

Como ya hemos comentado antes, nuestro estudio se basa en observar qué ventajas tiene el diseño trabajando con patrones y sin ellos.

Por tanto, y como hemos hecho en la primera fase, hemos tomado los tiempos que nos ha llevado realizar las actividades de la fase tercera, para compararla con la de la primera fase.

Estos tiempos son los siguientes:

- Realización de historias de usuarios: 45 minutos.
- Estimación de historias de usuarios: 20 minutos.
- Realización de casos de prueba: 1 hora 15 minutos.

En el siguiente apartado analizaremos los datos anteriores con los obtenidos en la fase primera, observando la eficiencia en tiempo y diseño de trabajar con los patrones de extreme programming.

13. ANALISIS DE LOS RESULTADOS OBTENIDOS

Como hemos mencionado previamente, nuestro estudio se basa en comprobar que gracias a los patrones de extreme programming descritos en la segunda fase, se llega a un diseño más eficiente que sin utilizar ningún patrón.

Para realizar este estudio se han tomado tiempos de desarrollo de las actividades, tanto para la primera fase, donde no se utilizaba ningún patrón, como para la tercera fase, donde se han aplicado los patrones de extreme programming descritos en la segunda fase.

Estos tiempos se han tomado en el desarrollo de las siguientes actividades:

- Tiempo de Realización de historias de usuario.
- Tiempo de Estimación de historias de usuario.
- Tiempo de Realización de casos de prueba de las historias de usuarios.

A continuación se expondrá una tabla aclaratoria de dichos tiempos para cada una de las dos fases.

- FASE 1: Desarrollo de actividades sin seguir ningún patrón.
- FASE 3: Desarrollo de actividades utilizando como fuente de conocimiento los patrones de extreme programming descritos en la fase segunda.

TIEMPOS	SIN PATRONES XP	CON PATRONES XP
Realización de Historias de Usuario	1h 45 min	45 min
Estimación de Historias de Usuario	45 min	20 min
Realización de Casos de Prueba	2h 45 min	1h 15 min

Nota: Los tiempos tomados son la suma del tiempo empleado en realizar todas las actividades por separado en cada una de las fases, es decir, por ejemplo, el tiempo en realizar las historias de usuario será la suma del tiempo empleado en realizar cada historia de usuario en cada una de las fases.
También hay que mencionar que el número de historias de usuario en la fase primera y tercera son comparables ya que aunque en la fase tercera hay una tarea más, la complejidad de estas actividades en la primera fase y en la tercera con semejantes, es decir, la

complejidad de realizar las historias de usuarios en la fase primera será semejante a la realización de las historias en la fase tercera.

Estos datos se pueden comprobar de una forma más visual gracias al gráfico que expondremos a continuación:

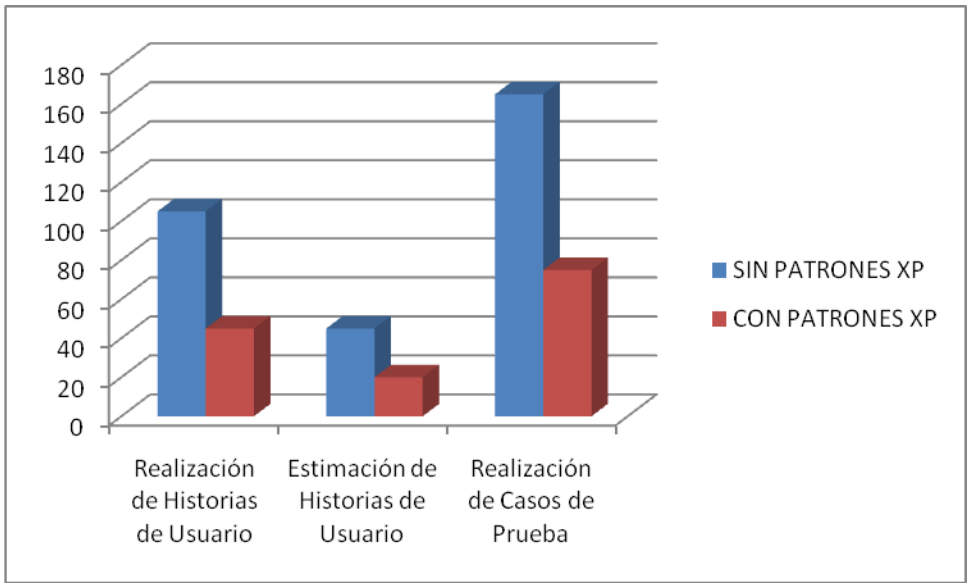


Figura 12: Diagrama de barras. Comparación tiempos fases 1 y 3

Nota: El tiempo expuesto en el diagrama de barras (eje de ordenadas) se ha medido en minutos.

Como podemos observar, en el desarrollo de todas y cada una de las actividades el tiempo de realización de las mismas utilizando como fuente de conocimiento los patrones extreme programming es menor que sin la utilización de los patrones.

Con esto lo que se quiere reflejar, es que gracias a la utilización de los patrones se ha llegado a una mayor eficiencia tanto en tiempo de ejecución como en diseño y aplicación de la metodología XP.

Por tanto, un desarrollador junior, tendrá mayor facilidad a la hora de aplicar esta metodología ágil teniendo como guía un conjunto de patrones que le dé las pautas para el desarrollo de su proyecto.

14. CONCLUSIONES

El objeto de estudio de este proyecto ha sido la comprobación de que gracias a la utilización de patrones en el desarrollo de software se puede llegar a tener una mayor eficiencia en el trabajo a desempeñar por los desarrolladores.

Estos patrones se generan para que dichos desarrolladores, en general desarrolladores junior y sin experiencia previa de la utilización de la metodología XP, puedan tener una mayor facilidad de comprensión y puedan tener algún tipo de guía para crear sus proyectos.

Además trabajar siguiendo estos artefactos beneficia a los clientes minimizando riesgos de los proyectos de desarrollo, aumentando su calidad y eficiencia.

Una de las partes que ha llevado más tiempo en la realización de este proyecto, ha sido la búsqueda de información acerca de la metodología XP ya que, en los libros de texto e internet encontrados acerca de este tema giran en torno a trabajos realizador por Kent Beck, lo que nos limita bastante el conocimiento, porque todos los documentos vienen a decir más o menos lo mismo aunque, al final hemos extraído la suficiente información como para poder realizar nuestro estudio de una forma normal.

En algún momento en la escritura del proyecto se ha comentado que, una de las partes más importantes de la metodología extreme programming es la reutilización. Pues bien, esta es la parte que ha costado más encontrar, ya que, prácticamente no hemos hallado ningún tipo de ayuda en este aspecto por parte de otros desarrolladores. Más bien, la documentación encontrada y utilizada para este proyecto se basa prácticamente en la explicación del método de una forma más general.

Además una vez analizados los datos y utilizada esta metodología deberemos comentar que la utilización de XP es una buena elección, ya que esta surgió como respuesta y posible solución a los problemas derivados del cambio en los requerimientos.

Además XP se plantea como una metodología a emplear en proyectos de riesgo y aumenta la productividad.

Una de las principales ventajas de XP es la agilidad que se confiere al proceso de software propuesto y su aplicación directa a proyectos pequeños con una tasa alta de volatilidad de requisitos.

No obstante, el propio Kent Beck sabe que si se intenta seguir esta metodología de forma inflexible, puede resultar que los proyectos terminen siendo, en algunos casos infructuosos. En este sentido Beck, nos llama hacia la agilidad como una forma de sacar nuestro sentido común, emplear nuestro saber hacer y no los procedimientos tipo que aplicar.

Por último, comentar que gracias a la elaboración de este proyecto se ha obtenido una idea de lo que las metodologías ágiles, y en concreto extreme programming , son capaces de generar, como por ejemplo, una mayor claridad, menores riesgos, trato más cercano al cliente, etc.

Adicionalmente, la aplicación de los patrones XP nos ha enseñado que siguiendo unas pequeñas pautas de cómo hacer el trabajo puede mejorar notablemente la eficiencia de nuestros proyectos además de la rapidez de ejecución de las actividades que este conlleva.

15. BIBLIOGRAFÍA

[1] Beck, K. *Una explicación de la programación extrema: Acepta el cambio*. Ed. Addisson Wesley. 2000.

[2] Martín, R. y Newkirk, J. *La programación Extrema en la práctica*. Ed. Addisson Wesley. 2001

[3] Álvarez, J. R. y Arias, M. *Método Extreme Programming*.
URL: http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html

[4] Anaya Villegas, A. *A propósito de programación extrema XP (extreme programming).*
URL: http://www.monografias.com

[5] De Seta, L. *Una introducción a Extreme Programming*.
URL: http://www.dosideas.com/noticias/metodologias/822-una-introduccion-a-extreme-programming.html

[6] *Extreme Programming: A gentle introduction*.
URL: http://www.extremeprogramming.org/

[7] Joskowicz, J. *Reglas y prácticas en Extreme Programming*.
URL: http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf

[8] Letelier, P. y Panadés M.C. *Metodologías Ágiles en el desarrollo de software: extreme programming*.
URL: http://www.willydev.net/descargas/masyxp.pdf

[9] Machado,F. *MemoriaDeGradoXP*.
URL: http://www.alejandrogoyen.com/MemoriaDeGradoXP.pdf

[10] Fernandez, G. (2002). *Introduccion a Extreme Programming*
URL: http://www.dsi.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo-XP.pdf